**TUGAS MANDIRI**

**FULL STACK PROJECT**

**Aplikasi *Social Media* dan Aplikasi *Tracker Expenses***



|  |  |  |
| --- | --- | --- |
| Nama | : | Alvina Leony |
| NPM | : | 221510003 |
| Dosen | : | Saut Pintubipar Saragih, S.Kom., M.MSI. |

**PROGRAM STUDI SISTEM INFORMASI FAKULTAS TEKNIK DAN KOMPUTER UNIVERSITAS PUTERA BATAM**

**2024**

# 

# DAFTAR ISI

[DAFTAR ISI ii](#_Toc169955347)

[DAFTAR GAMBAR iv](#_Toc169955348)

[BAB I 6](#_Toc169955349)

[1. 1 Social Media 6](#_Toc169955350)

[1.1.1 Latar Belakang 6](#_Toc169955351)

[1.1.2 Validasi 7](#_Toc169955352)

[1.2 Tracker Expanses 8](#_Toc169955353)

[1.2.1 Latar Belakang 8](#_Toc169955354)

[1.2.2 Validasi 9](#_Toc169955355)

[BAB II 10](#_Toc169955356)

[TINJAUAN PUSTAKA 10](#_Toc169955357)

[2.1 Kajian Teori 10](#_Toc169955358)

[2.1.1 Visual Code 10](#_Toc169955359)

[2.1.2 MERN 10](#_Toc169955360)

[2.1.3 Manajer paket Yarn 13](#_Toc169955361)

[2.1.4 Babel 13](#_Toc169955362)

[2.1.5 Webpack 13](#_Toc169955363)

[2.1.6 Nodemon 13](#_Toc169955364)

[2.1.7 FIGMA 14](#_Toc169955365)

[BAB III 15](#_Toc169955366)

[3.1 Aplikasi Sosial Media 15](#_Toc169955367)

[3.1.1 Hasil Aplikasi 15](#_Toc169955368)

[3.1.1.1 Halaman Utama 15](#_Toc169955369)

[3.1.1.2 Halaman Sign Up 15](#_Toc169955370)

[3.1.1.3 Halaman Sign In 16](#_Toc169955371)

[3.1.1.4 Halaman Beranda 16](#_Toc169955372)

[3.1.1.4 Halaman Profile 17](#_Toc169955373)

[3.1.2 Pembahasan 17](#_Toc169955374)

[3.1.2.1 UI/UX 17](#_Toc169955375)

[3.1.2.2 Coding 25](#_Toc169955376)

[A. Halaman Utama 25](#_Toc169955377)

[B. Sign Up (Daftar) 30](#_Toc169955378)

[C. Sign In (Masuk) 36](#_Toc169955379)

[D. Sign Out (Keluar) 45](#_Toc169955380)

[E. My Profile 47](#_Toc169955381)

[F. Halaman Beranda 69](#_Toc169955382)

[3.2 Aplikasi Expenses Tracker 118](#_Toc169955383)

[3.2.1 Hasil Aplikasi 118](#_Toc169955384)

[3.2.1.1 Halaman Utama 118](#_Toc169955385)

[3.2.1.2 Halaman Sign Up 119](#_Toc169955386)

[3.2.1.3 Halaman Sign In 119](#_Toc169955387)

[3.2.1.4 Halaman My Profile 120](#_Toc169955388)

[3.2.1.5 Halaman Expenses 120](#_Toc169955389)

[3.2.1.6 Halaman Add Expenses 121](#_Toc169955390)

[3.2.1.1 Halaman Reports Menu 121](#_Toc169955391)

[3.2.2 Pembahasan 122](#_Toc169955392)

[3.2.2.1 UI/UX 122](#_Toc169955393)

[3.2.2.2 Coding 129](#_Toc169955394)

[A. Halaman Utama 129](#_Toc169955395)

[B. Sign Up (Daftar) 134](#_Toc169955396)

[C. Sign In (Masuk) 141](#_Toc169955397)

[D. Sign Out (Keluar) 150](#_Toc169955398)

[E. Sign Out (Keluar) 152](#_Toc169955399)

[F. Reports Menu 177](#_Toc169955400)

[G. Expenses scattered over 180](#_Toc169955401)

[H. Your monthly expenditures 187](#_Toc169955402)

[I. Expenditures per category 193](#_Toc169955403)

[J. Add Expense button 201](#_Toc169955404)

[J. My Profile Menu 209](#_Toc169955405)

[BAB IV 222](#_Toc169955406)

[SIMPULAN 222](#_Toc169955407)

[DAFTAR PUSTAKA 223](#_Toc169955408)

# DAFTAR GAMBAR

[Gambar 1.1 Validasi pertanyaan 1 pada sosial media 7](#_Toc169955409)

[Gambar 1.2 Validasi pertanyaan 2 pada sosial media 7](#_Toc169955410)

[Gambar 1.3 Validasi pertanyaan 1 pada Tracker Expenses 9](#_Toc169955411)

[Gambar 1.4 Validasi pertanyaan 2 pada Tracker Expenses 9](#_Toc169955412)

[Gambar 2.1 Halaman Utama Sosial Meida 15](#_Toc169955413)

[Gambar 2.2 Halaman Sign Up Sosial Meida 15](#_Toc169955414)

[Gambar 2.3 Halaman Sign Ip Sosial Meida 16](#_Toc169955415)

[Gambar 2.4 Halaman Beranda Sosial Meida 16](#_Toc169955416)

[Gambar 2.5 Halaman My Profile Sosial Meida 17](#_Toc169955417)

[Gambar 2.6 Halaman Utama (UI/UX) 17](#_Toc169955418)

[Gambar 2.7 Halaman Sign Up Sosial Meida (UI/UX) 18](#_Toc169955419)

[Gambar 2.8 Nama, Email, Password 18](#_Toc169955420)

[Gambar 2.9 Button Sign Up 19](#_Toc169955421)

[Gambar 2.10 Button Submit 19](#_Toc169955422)

[Gambar 2.11 Halaman Sign In Sosial Meida (UI/UX) 19](#_Toc169955423)

[Gambar 2.12 Button Sign In 20](#_Toc169955424)

[Gambar 2.13 Button Submit 20](#_Toc169955425)

[Gambar 2.14 Halaman Beranda Sosial Media 20](#_Toc169955426)

[Gambar 2.15 Button Sign Out 21](#_Toc169955427)

[Gambar 2.16 Button Post 21](#_Toc169955428)

[Gambar 2.17 Ikon Camera 21](#_Toc169955429)

[Gambar 2.18 Ikon Trash 21](#_Toc169955430)

[Gambar 2.19 Ikon Like 22](#_Toc169955431)

[Gambar 2.20 Comment 22](#_Toc169955432)

[Gambar 2.21 Delete Comment 22](#_Toc169955433)

[Gambar 2.22 Who to follow 22](#_Toc169955434)

[Gambar 2.23 Ikon See 23](#_Toc169955435)

[Gambar 2.24 Follow 23](#_Toc169955436)

[Gambar 2.25 Unfollow 23](#_Toc169955437)

[Gambar 2.26 Follow & Unfollow Button 23](#_Toc169955438)

[Gambar 2.27 Halaman My Profile 24](#_Toc169955439)

[Gambar 2.28 Ikon Edit & Delete 24](#_Toc169955440)

[Gambar 2.29 Tab Posts, Following, Followers 24](#_Toc169955441)

[Gambar 2.30 Ikon Home 25](#_Toc169955442)

[Gambar 2.31 Halaman Utama Expense Tracker 118](#_Toc169955443)

[Gambar 2.32 Halaman Sign Up Expense Tracler 119](#_Toc169955444)

[Gambar 2.32 Halaman Sign In Expense Tracker 119](#_Toc169955445)

[Gambar 2.33 Halaman My Profile Expense Tracker 120](#_Toc169955446)

[Gambar 2.34 Halaman Expense Tracker 120](#_Toc169955447)

[Gambar 2.35 Halaman Add Expense 121](#_Toc169955448)

[Gambar 2.36 Halaman Reports Menu 121](#_Toc169955449)

[Gambar 2.37 Halaman Utama 122](#_Toc169955450)

[Gambar 2.37 Halaman Sign Up 123](#_Toc169955451)

[Gambar 2.38 Halaman Sign In 123](#_Toc169955452)

[Gambar 2.39 Halaman Sesudah Sign In 124](#_Toc169955453)

[Gambar 2.40 Expenses Button 124](#_Toc169955454)

[Gambar 2.41 Reports Button 125](#_Toc169955455)

[Gambar 2.42 Halaman Beranda 125](#_Toc169955456)

[Gambar 2.43 Diagram Pie 126](#_Toc169955457)

[Gambar 2.44 Diagram Bar 127](#_Toc169955458)

[Gambar 2.45 Category Expenses 128](#_Toc169955459)

[Gambar 2.46 Add Expense Button 128](#_Toc169955460)

[Gambar 2.47 Halaman My Profile 128](#_Toc169955461)

[Gambar 2.48 Edit Button 129](#_Toc169955462)

[Gambar 2.49 Trash Button 129](#_Toc169955463)

# 

# BAB I

## 1. 1 Social Media

### 1.1.1 Latar Belakang

Media sosial merupakan media online yang memudahkan para penggunanya saling

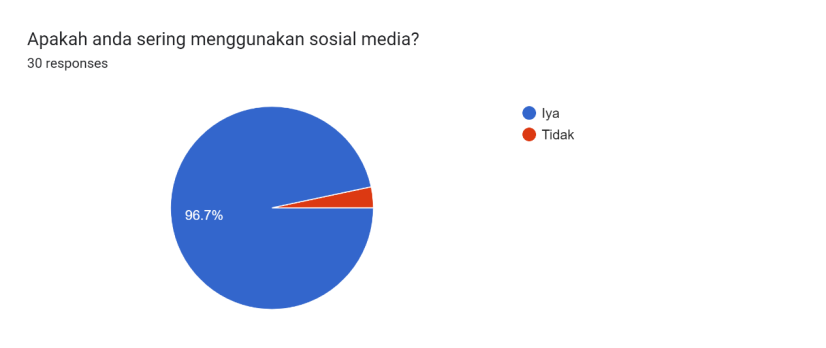
berkomunikasi, menciptakan dan berbagi pesan antar pengguna yang terhubung melalui jaringan media online (Muhammad Iqbal Sultan, 2020). Dalam era digital saat ini, sosial media telah menjadi bagian tak terpisahkan dari kehidupan sehari-hari, membentuk landskap komunikasi dan interaksi sosial yang baru. Keberadaan sosial media telah merata di berbagai lapisan masyarakat, dari anak-anak hingga orang dewasa, dan bahkan bisnis besar hingga usaha kecil. Dengan teknologi yang makin canggih, penggunaan social menjadi meningkat, tetapi dengan adanya social media ini kita dapat berkomunikasi dengan jarak jauh serta dapat menambah teman tanpa harus bertemu secara langsung.

Media sosial mengacu pada penggunaan teknologi berbasis web guna mengubah komunikasi menjadi dialog interaktif (Muhamad Ayub, n.d.). Pentingnya sebuah proyek sosial media, terutama dalam bentuk kelompok atau komunitas, menjadi semakin relevan karena kemampuannya untuk menghubungkan orang-orang dengan minat, tujuan, atau nilai-nilai bersama. Proyek-proyek ini mampu memfasilitasi kolaborasi, pertukaran ide, dan dukungan antaranggota dalam mencapai tujuan bersama. Perancangan aplikasi dengan menggunakan MERN yang memberikan banyak pengetahuan dan manfaat dari MERN itu sendiri. Dalam konteks bisnis lebih dari sekadar alat komunikasi, proyek sosial media yang sukses dapat menjadi wadah bagi pertukaran budaya, penyebaran pengetahuan, dan pembentukan identitas online.

Penelitian menunjukkan bahwa pengguna media sosial sangat banyak dengan ini menggunakan fitur-fitur yang memfasilitasi interaksi sosial dan pengelolaan identitas digital mereka. Berdasarkan penelitian ini, pengembangan aplikasi media sosial yang sukses akan memperhatikan kebutuhan dan preferensi pengguna terhadap fitur-fitur ini. Seperti fitur delete yang merupakan salah satu fitur yang berguna bagi pengguna social media ini.

Fitur delete account sangat berguna bagi pengguna yang ingin menghentikan penggunaan platform untuk alasan privasi, keamanan, atau hanya karena mereka ingin mengurangi keberadaan online mereka. Pengguna mungkin merasa perlu menghapus akun mereka jika mereka khawatir tentang pelanggaran data, atau jika mereka merasa bahwa mereka tidak lagi ingin berpartisipasi dalam komunitas tertentu. Dengan kemampuan untuk menghapus akun secara permanen, pengguna dapat merasa lebih aman dan memiliki kendali lebih besar atas data pribadi mereka.

### 1.1.2 Validasi

****

Gambar 1.1 Validasi pertanyaan 1 pada sosial media

Indonesia dengan total populasi 8.01 milliar, populasi yang menggunakan internet terdapat 5.16 milliar, dan yang aktif social media mencapati 4.76 milliar, dengan setiap tahunnya bertambah 3.0 % (We Are Social Digital 2023, n.d.).



Gambar 1.2 Validasi pertanyaan 2 pada sosial media

Survei Sprout Pulse Q4 2023 terhadap 1.623 konsumen, 65% pelanggan menghubungi merek di media sosial dengan pertanyaan tentang produk dan layanan sebelum melakukan pembelian. Cara merespons dapat memengaruhi potensi penjualan. Komentar-komentar pada social media ini penting untuk melacak sentimen merek sehingga anda tahu bagaimana audiens mempersepsikan merek anda (Annette Chacko, 2023).

## 1.2 Tracker Expanses

### 1.2.1 Latar Belakang

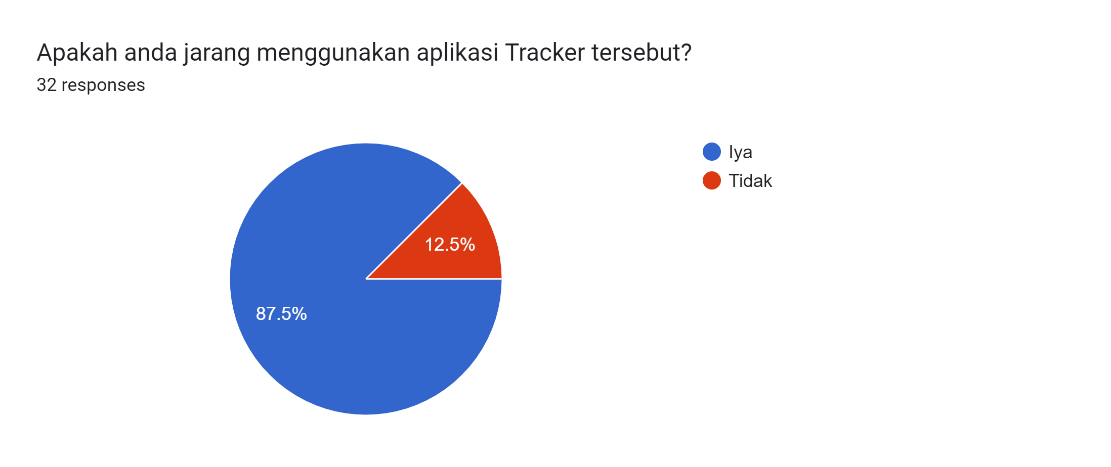
Aplikasi Tracker Expenses adalah alat atau aplikasi yang digunakan untuk mencatat, memantau, dan menganalisis pengeluaran seseorang atau suatu kelompok dalam rentang waktu tertentu. Aplikasi Pelacak Pengeluaran ini termasuk dalam kategori Keuangan dan digunakan untuk mengelola dana, yang merupakan aspek penting dalam kehidupan seseorang (Nidhi Jitendra Jadhav, 2022).

Kegunaannya sangat signifikan dalam mengelola keuangan karena membantu pengguna untuk melacak uang yang dikeluarkan, menganalisis pola pengeluaran, dan membuat anggaran yang lebih efektif. Pengguna dapat lebih memahami pengeluaran mereka berkat fitur distribusi berdasarkan kategori, yang memudahkan mereka untuk tetap berpegang pada anggaran mereka (Asst.Prof. Abhishek Dadhich, 2023). Keberadaan aplikasi Tracker Expenses tidak hanya bermanfaat bagi individu, tetapi juga bagi tim atau kelompok yang memiliki kebutuhan untuk mengelola keuangan bersama-sama.

Dalam proyek aplikasi ini memungkinkan tim untuk berkomunikasi tentang kemajuan proyek, memecahkan masalah yang muncul, dan menyinkronkan upaya mereka untuk mencapai tujuan bersama. Perancangan aplikasi dengan menggunakan MERN yang memberikan banyak pengetahuan dan manfaat dari MERN itu sendiri. Kolaborasi yang baik antara anggota tim menjadi kunci utama dalam menjaga kelancaran proyek dan mencapai visi yang telah ditetapkan. Tren meningkatnya kesadaran finansial dan kompleksitas kehidupan keuangan menjadi pendorong utama dalam permintaan akan solusi manajemen keuangan yang lebih efektif. Dengan kemajuan teknologi mobile dan pengembangan perangkat lunak, peluang baru terbuka untuk menciptakan aplikasi Tracker Expenses yang mudah digunakan dan dapat diakses di mana saja.

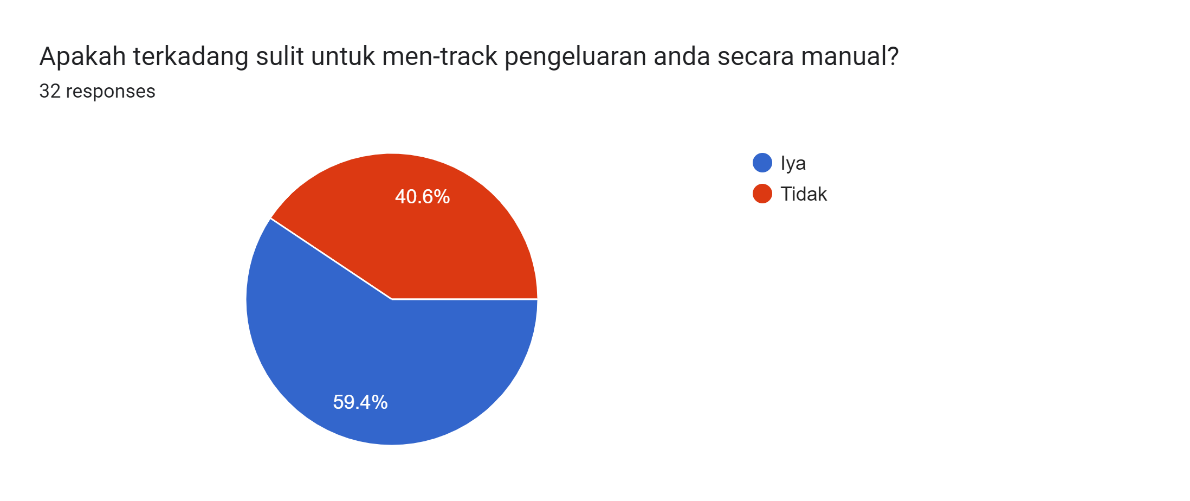
Proses login yang rumit atau memakan waktu dapat menghambat akses pengguna ke aplikasi dan mengurangi tingkat keterlibatan mereka. Dengan menyediakan opsi login yang mudah dan cepat, aplikasi dapat meningkatkan kenyamanan pengguna dan mempercepat akses ke data mereka. Fitur-fitur yang mendukung untuk memudahkan pengguna dalam memakai aplikasi memberikan manfaat yang baik untuk pengguna. Seperti fitur laporan adalah elemen penting dalam aplikasi tracker karena memungkinkan pengguna untuk menganalisis dan memahami data mereka dengan lebih baik. Pertanggalan juga tidak boleh diabaikan dalam pengembangan aplikasi tracker.

### 1.2.2 Validasi



Gambar 1.3 Validasi pertanyaan 1 pada Tracker Expenses

Survey yang dilakukan pada orang di India yang dilakukan pada April tahun 2020, dengan total responden 452 semasa lockdown saat corona. Dari hasil survey tersebut, banyak yang tidak menggunakan aplikasi tracker expenses ini, dikarenakan saat menggunakan tracker expenses ini mereka harus mencatat secara manual sehinggak jarang menggunakannya. Berdasarkan survey ini, banyak yang pernah menggunakan aplikasi ini tapi jarang yang bertahan lama (D91 Labs, 2020).



Gambar 1.4 Validasi pertanyaan 2 pada Tracker Expenses

Survey yang dilakukan mendapatkan hasil, banyak orang yang terkadang mengalami kesulitan untuk selalu men-track pengeluaran mereka dengan aplikasi tracker ini.= (D91 Labs, 2020).

# BAB II

# TINJAUAN PUSTAKA

## 2.1 Kajian Teori

### 2.1.1 Visual Code

Visual Studio Code (VS Code) ini adalah sebuah teks editor ringan dan hl yang dibuat oleh Microsoft untuk sistem operasi multiplatform, artinya tersedia juga untuk versi Linux, Mac, danWindows. Teks editor ini secara langsung mendukung bahasa pemrograman JavaScript, Typescript, dan Node.js, serta bahasa pemrograman lainnya dengan bantuan plugin yang dapat dipasang via marketplace Visual Studio Code. Penggunaan Visual Studio Code sebagai editor pengembangan memberikan pengalaman praktis yang mendalam dalam pembuatan web sederhana (seperti C++, C#, Python, Go, Java, dst)(Abdur Ra’uf Al Farras1, 2023).

Kelebihan utama dari Visual Studio Code adalah antarmuka pengguna yang bersih dan mudah dimengerti. Kode bisa dilihat di sana, serta disusun dengan rapi dalam berbagai file dan folder. Selain itu, Visual Studio Code juga dilengkapi dengan banyak fitur berguna, seperti penyorotan sintaksis (mengubah warna kata-kata penting dalam kode), otomatisasi (membantu menulis kode lebih cepat), dan debugging (membantu menemukan dan memperbaiki kesalahan dalam kode). Visual Studio Code sangat populer adalah ekstensibilitasnya. Fitur-fitur tambahan bisa ditambahkan dengan menginstal ekstensi dari toko ekstensi bawaan. Misalnya, dukungan untuk bahasa pemrograman tertentu atau alat bantu lainnya sesuai dengan kebutuhan.

### 2.1.2 MERN

MERN stack terdiri dari empat teknologi utama: MongoDB, Express, React, dan Node.js (Shama Hoque, 2018). MERN stack menggabungkan empat teknologi utama yang bekerja bersama untuk membangun aplikasi web secara efisien. Node.js dan Express, yang menyediakan fondasi kuat untuk server dan routing aplikasi. Kemudian, MongoDB sebagai penyimpanan data yang fleksibel dan dapat diakses. Dari sisi antarmuka pengguna, React menjadi kunci, memungkinkan kita untuk dengan mudah membuat tampilan yang dinamis dan interaktif. React dibangun di sekitar objek, bukan template seperti kerangka kerja lainnya (Monika Mehra, 2021).

Masing-masing teknologi dalam MERN stack memiliki peran uniknya sendiri, tetapi ketika digabungkan bersama, mereka membentuk kerangka kerja yang kuat untuk pengembangan web. Jadi, saat membangun aplikasi web menggunakan MERN stack, yang akan digunakan adalah MongoDB untuk menyimpan data, Express untuk mengatur bagaimana data tersebut dikirimkan dari server ke browser, React untuk membuat antarmuka pengguna, dan Node.js untuk menjalankan server web, dan semuanya menggunakan bahasa JavaScript, jadi tidak perlu belajar bahasa baru, yang perlu difokuskan ialah memahami cara menggunakan setiap alat dengan baik.

#### A. Mango

MongoDB adalah salah satu sistem manajemen basis data NoSQL yang memungkinkan pengguna untuk melakukan pencarian data dalam berbagai bidang menggunakan dokumen sebagai basisnya (Tessalonica Putry Avrylya, 2024). Dengan arsitektur terdistribusi, MongoDB mendukung ketersediaan tinggi dan skalabilitas horizontal. Bahasa kueri yang ekspresif memungkinkan kueri ad hoc dan agregasi real-time. Memilih MongoDB untuk aplikasi web Node dan Express memberikan aplikasi sisi server yang berbasis JavaScript dan independen, memungkinkan integrasi dengan antarmuka sisi klien menggunakan React. Sebelum kita bisa menggunakan database pada aplikasi MERN, seperti MongoDB, kita harus pastikan bahwa MongoDB sudah diinstal dan berjalan dengan baik. Setelah MongoDB siap, langkah berikutnya adalah menggunakan alat yang disebut mongo shell. Ini adalah semacam "pintu masuk" ke MongoDB yang memungkinkan kita untuk melakukan banyak hal, seperti mencari data, mengubah data, atau melakukan tugas-tugas administratif langsung pada database. Jadi, dengan menggunakan mongo shell, kita bisa mengendalikan dan mengelola database kita dengan mudah dan cepat menggunakan perintah-perintah sederhana di baris perintah (Shama Hoque, 2018).

#### B. Express

Express adalah kerangka kerja sederhana untuk sisi server yang digunakan untuk membangun aplikasi web dengan Node. Ini menambahkan lapisan fitur aplikasi web dasar ke Node yang menyediakan metode utilitas HTTP dan fungsionalitas middleware. Middleware adalah sejenis "perantara" dalam sebuah aplikasi yang memungkinkan berbagai komponen bekerja sama. Dalam aplikasi web sisi server, seperti yang dibangun dengan Node.js, Express adalah kerangka kerja yang umum digunakan untuk menangani routing dan middleware. Backend dibangun oleh Node.js dan framework Express (Mohammad Shahanewaz Shahabuddin, 2023).

Dalam aplikasi MERN, yang menggunakan Express di sisi server, Express berfungsi untuk menangani banyak tugas seperti routing, menyajikan file statis kepada pengguna, mengelola akses ke data melalui otentikasi, dan menambahkan fungsi tambahan lainnya yang diperlukan. Jadi, Express dan middleware-nya memberikan fleksibilitas yang besar dalam membangun aplikasi web yang sesuai dengan kebutuhan. Fungsi penting dalam setiap aplikasi web lengkap adalah sistem penyimpanan data. Modul Express tidak menetapkan persyaratan atau memberlakukan pembatasan pada integrasi database dengan aplikasi web Node-Express. Oleh karena itu, ini memberi fleksibilitas untuk memilih opsi database apa pun, baik itu database relasional seperti PostgreSQL atau database NoSQL seperti MongoDB.

#### C. React

React terutama digunakan untuk mendorong sistem JavaScript sisi pelanggan (Mohanish Bawane1, 2022). Ini membuat pengembang lebih mudah dalam membuat dan merawat antarmuka pengguna yang dapat digunakan kembali, interaktif, dan kompleks karena sifatnya yang deklaratif dan berbasis komponen. Aplikasi besar yang menampilkan banyak data yang berubah-ubah dapat cepat dan responsif jika dibangun dengan React, karena React mengatur pembaruan dan penggambaran komponen antarmuka pengguna dengan efisien ketika data tertentu berubah. React melakukan penggambaran ini dengan menggunakan virtual DOM, yang membedakannya dari perpustakaan antarmuka pengguna web lainnya yang menangani pembaruan halaman dengan manipulasi langsung di DOM browser yang mahal. Mengembangkan antarmuka pengguna menggunakan React juga memaksa para programmer frontend untuk menulis kode yang modular, dapat digunakan kembali, dan lebih mudah untuk dibug, diuji, dan diperluas.

#### D. Node

Node dikembangkan sebagai lingkungan runtime JavaScript yang dibangun di atas mesin JavaScript V8 milik Chrome (Basarat Syed, n.d.). Dengan adanya Node, JavaScript bisa digunakan di sisi server untuk membangun berbagai macam alat dan aplikasi, tidak hanya terbatas pada penggunaan di dalam browser seperti sebelumnya (Andrew Mead, 2018). Sistem manajemen paket default Node, yaitu Node Package Manager atau npm, disertakan dalam instalasi Node. npm memberikan akses ke ratusan ribu paket Node yang dapat digunakan kembali yang dibangun oleh pengembang dari seluruh dunia. npm juga merupakan ekosistem perpustakaan open source terbesar di dunia saat ini.

Npm bukan satu-satunya sistem manajemen paket yang tersedia. Node akan memungkinkan kita membangun dan menjalankan aplikasi JavaScript full-stack lengkap. Namun, untuk mengimplementasikan aplikasi sisi server yang dapat diperluas dengan fitur-fitur khusus aplikasi web seperti routing API, kita akan menggunakan modul Express di atas Node. Implementasi backend server untuk aplikasi MERN bergantung pada Node.js. Instalasi Node.js dapat dilakukan melalui unduhan langsung, installer, atau dengan menggunakan Node Version Manager (nvm). Dengan Node.js sudah diatur di sistem, kita dapat menggunakan manajer paket seperti npm atau Yarn untuk mulai mengintegrasikan bagian-bagian lain dari tumpukan MERN.

### 2.1.3 Manajer paket Yarn

Yarn adalah manajer paket yang relatif baru untuk dependensi JavaScript, dan dapat digunakan sebagai alternatif untuk npm. Ini adalah manajer dependensi yang cepat, l, dan aman yang menyediakan berbagai fitur tambahan, termasuk mode offline untuk penginstalan ulang paket tanpa koneksi internet dan dukungan untuk beberapa registri paket, seperti npmjs.com dan Bower. Setelah Yarn diinstal, kita dapat menggunakannya untuk menambahkan modul-modul lain yang diperlukan termasuk Express dan React.

### 2.1.4 Babel

Babel adalah sebuah compiler JavaScript yang memungkinkan menulis kode JavaScript modern dan kemudian mengonversinya agar kompatibel dengan browser yang lebih lama(Ojaswi Awasthi, n.d.). Babel menjembatani celah antara JavaScript terbaru dan browser yang mungkin digunakan oleh pengguna kita. Babel memiliki beberapa kegunaan penting dalam pengembangan perangkat lunak JavaScript. Pertama, alat ini menyediakan kompatibilitas yang luas dengan berbagai browser. Ini berarti kode JavaScript yang ditulis menggunakan fitur terbaru dapat tetap berjalan dengan lancar di browser-browsers yang mungkin tidak mendukung fitur-fitur tersebut. Kedua, Babel membantu meningkatkan keterbacaan kode. Dengan demikian, Babel menjadi alat yang sangat berharga dalam meningkatkan produktivitas dan kualitas pengembangan perangkat lunak JavaScript.

### 2.1.5 Webpack

Webpack adalah alat bundler JavaScript yang memiliki berbagai kegunaan penting dalam pengembangan perangkat lunak. Webpack dapat menangani JavaScript serta entitas frontend lainnya seperti HTML, CSS, visual, dan media lainnya (Chakradhar Avinash Devarapalli, 2020). Alat ini membantu menggabungkan berbagai file JavaScript, CSS, dan aset lainnya menjadi satu file yang dioptimalkan untuk browser. Selain itu, Webpack juga mampu melakukan tugas-tugas tambahan, seperti minifikasi, transpilasi, dan pemuatan kode secara efisien. Kegunaan utama Webpack termasuk mengoptimalkan performa aplikasi dengan menggabungkan dan mengoptimalkan file JavaScript, yang secara langsung dapat meningkatkan kecepatan pemuatan halaman dan performa keseluruhan aplikasi. Selain itu, alat ini memudahkan pengelolaan kode JavaScript dengan menyederhanakan struktur proyek dan membantu dalam pengaturan kode.

### 2.1.6 Nodemon

Nodemon adalah alat pengembangan Node.js yang secara otomatis memantau perubahan file dalam direktori dan menjalankan ulang server Node.js setiap kali perubahan terdeteksi (Shama Hoque, 2018) . Hal ini menghemat waktu dan tenaga, karena tidak perlu terus-menerus menghentikan dan memulai server secara manual Nodemon menghemat waktu dengan secara otomatis me-restart server setiap kali ada perubahan dalam kode. Hal ini memungkinkan pengembang untuk fokus pada pengembangan kode tanpa harus membuang waktu untuk memulai ulang server secara manual. Nodemon mempermudah proses debugging tanpa perlu me-restart server secara manual. Ini membuat proses debugging menjadi lebih cepat dan efisien. Nodemon meningkatkan produktivitas dengan menghilangkan tugas-tugas manual yang berulang sehingga dapat bekerja dengan lebih efisien.

### 2.1.7 FIGMA

Figma adalah aplikasi desain berbasis cloud dan alat prototyping untuk proyek digital. Figma dibuat untuk dapat membantu para penggunanya agar bisa berkolaborasi dalam proyek dan bekerja dalam bentuk tim sekaligus di mana saja (Rully Pramudita, 2021).

Alasan-alasan mendesain UI/UX menggunakan Figma:

**1. Kolaborasi Real-Time**

Figma memungkinkan tim untuk bekerja sama secara real-time pada project yang sama. Hal ini berarti semua anggota tim dapat melihat dan mengedit desain secara bersamaan, sehingga meningkatkan komunikasi dan produktivitas.

**2. Prototyping Interaktif**

Figma memungkinkan desainer untuk membuat prototipe interaktif yang realistis. Prototipe ini dapat digunakan untuk mensimulasikan bagaimana tampilan dan cara kerja desain ketika digunakan, sehingga membantu tim untuk menguji desain dengan pengguna dan mendapatkan umpan balik, mengidentifikasi dan memperbaiki masalah sebelum desain difinalisasi, membuat presentasi desain yang menarik kepada stakeholder.

**3. Aksesibilitas**

Figma dapat diakses dari **berbagai perangkat** melalui browser. Hal ini memungkinkan tim untuk bekerja dari mana saja dan memastikan kontrol versi yang optimal.

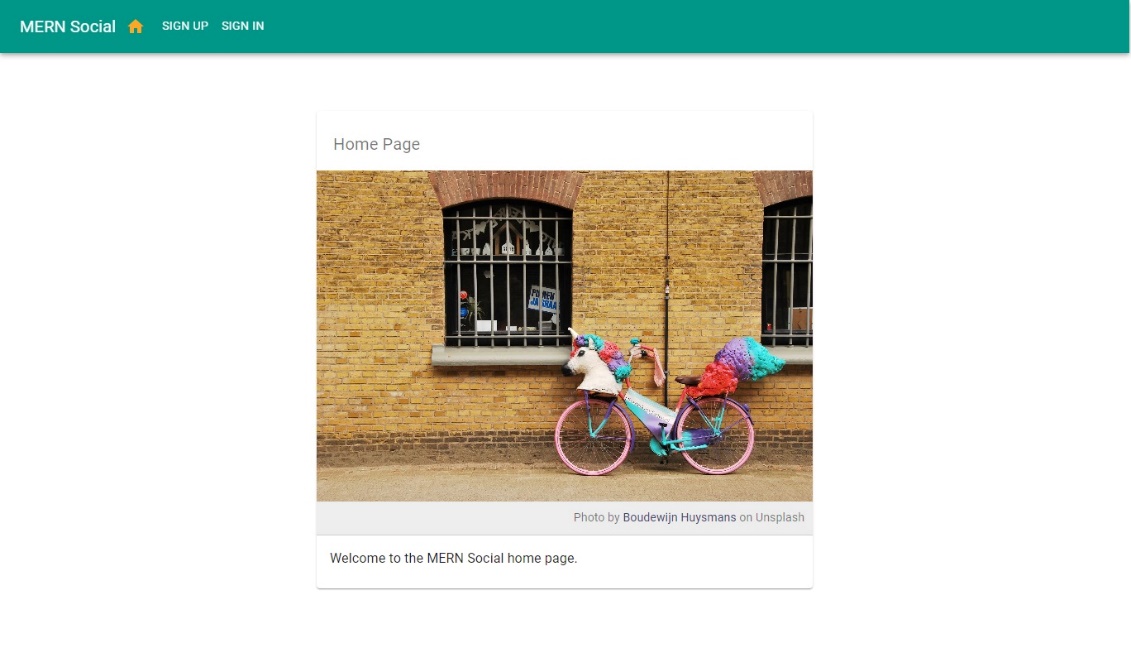
# BAB III

# 3.1 Aplikasi Sosial Media

## 3.1.1 Hasil Aplikasi

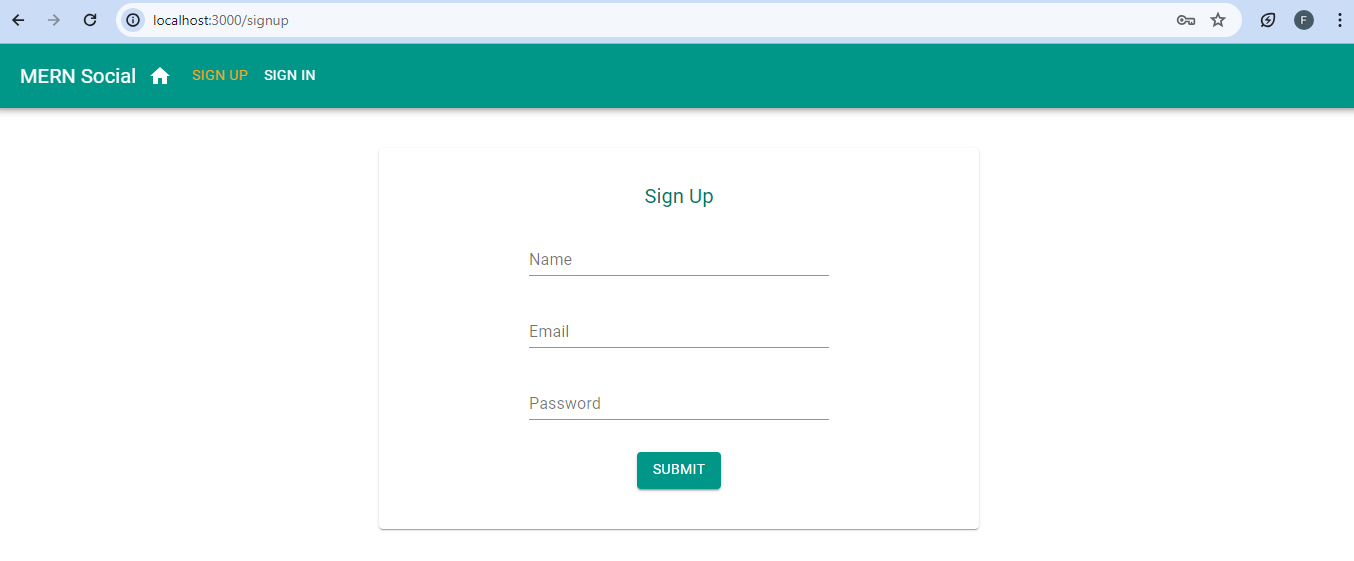
Berikut ini merupkan tampilan-tampilan pada aplikasi sosial media.

### Halaman Utama



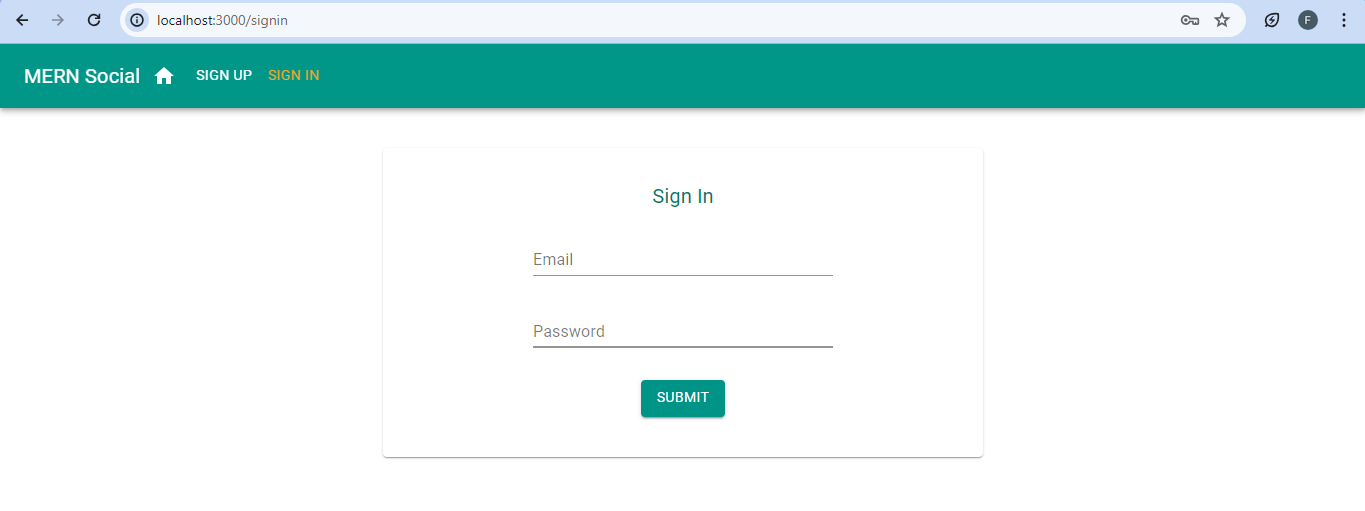
Gambar 2.1 Halaman Utama Sosial Meida

### Halaman Sign Up



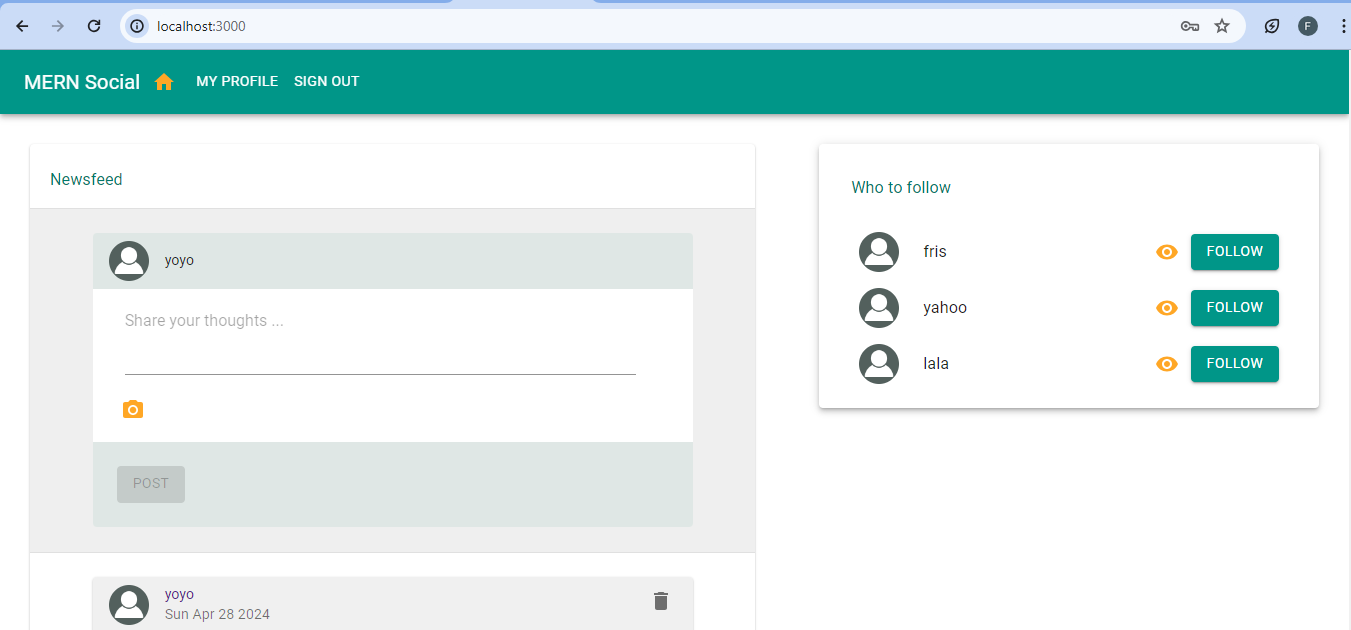
Gambar 2.2 Halaman Sign Up Sosial Meida

### Halaman Sign In



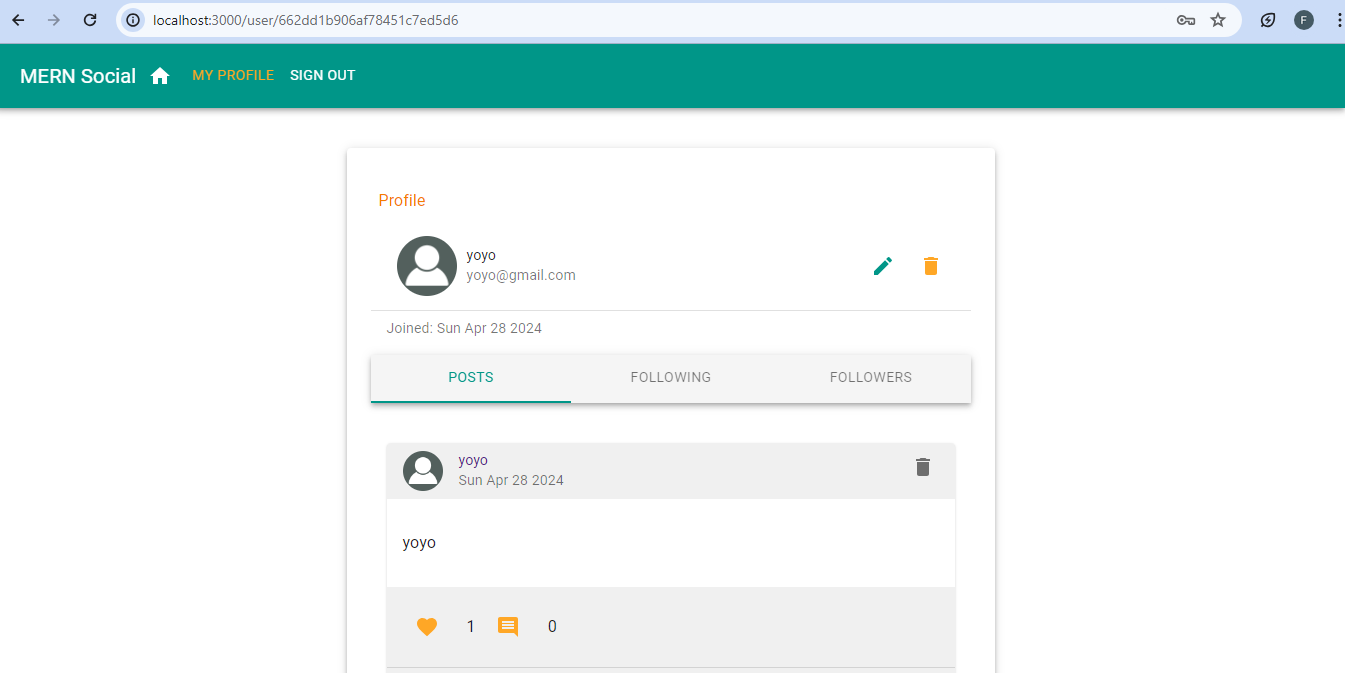
Gambar 2.3 Halaman Sign Ip Sosial Meida

### 3.1.1.4 Halaman Beranda



Gambar 2.4 Halaman Beranda Sosial Meida

### Halaman Profile

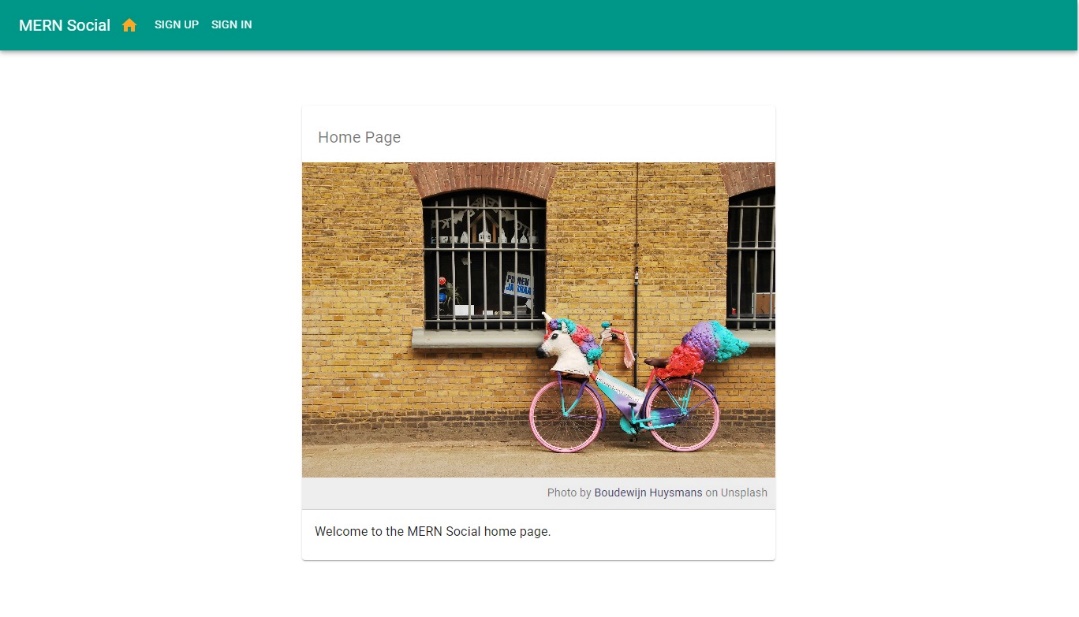


Gambar 2.5 Halaman My Profile Sosial Meida

## 3.1.2 Pembahasan

## 3.1.2.1 UI/UX

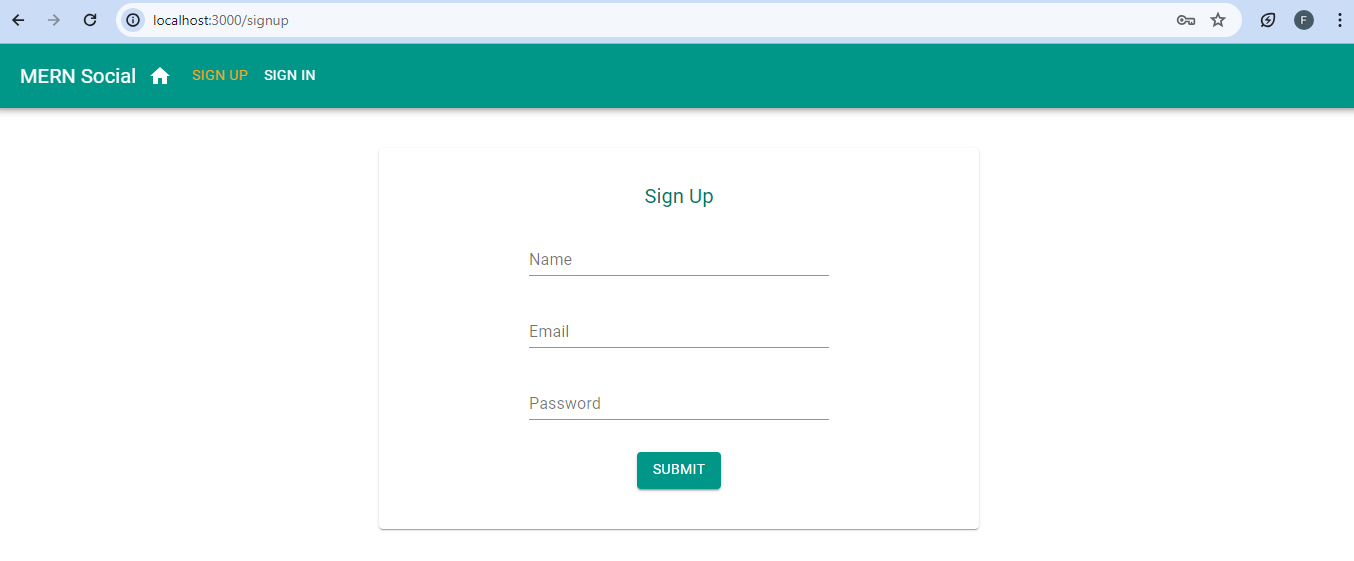
#### A. Halaman Utama



Gambar 2.6 Halaman Utama (UI/UX)

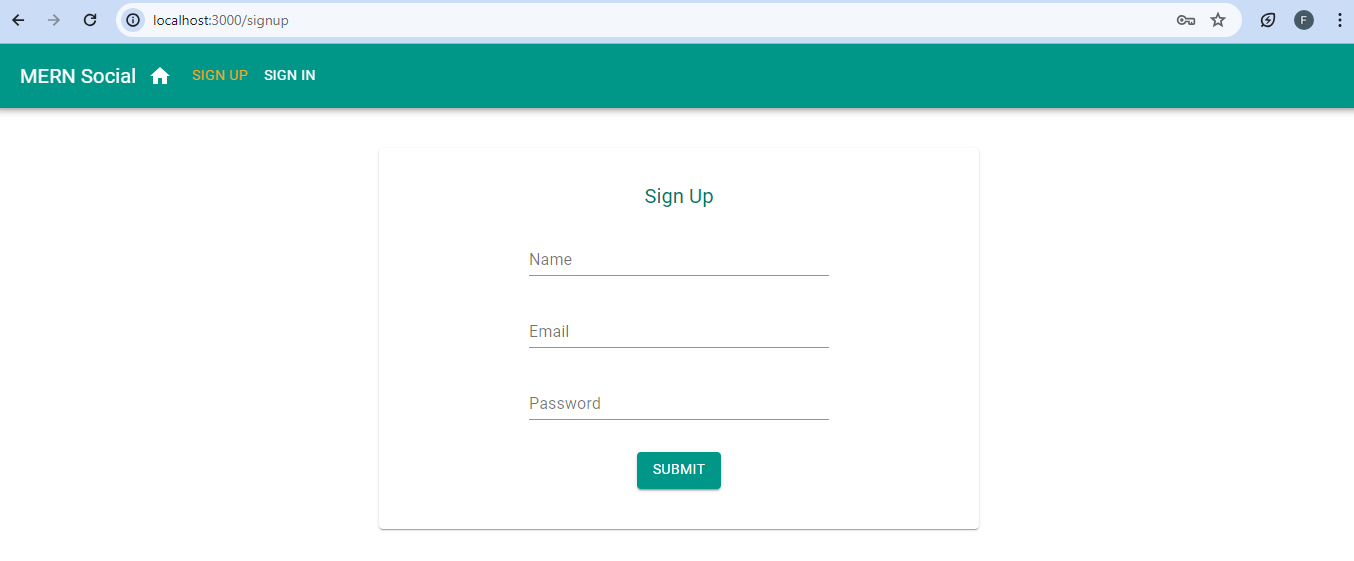
Gambar ini adalah halaman utama yang akan muncul saat membuka aplikasi media sosial sebelum memasukkan akun kita. Halaman ini menampilkan fitur sign up dan sign in yang memungkinkan kita masuk ke akun kita dan menggunakan media sosial tersebut. Teks "Selamat datang di halaman utama MERN Social" terletak di bawah fitur Sign Up dan Sign In untuk menyambut pengguna ke halaman beranda aplikasi MERN Social. Serta, foto Sosial Media Foto tersebut dimaksudkan untuk menunjukkan bahwa MERN Social adalah aplikasi untuk terhubung dengan orang lain dan membangun hubungan sosial.

#### B. Halaman Sign Up



Gambar 2.7 Halaman Sign Up Sosial Meida (UI/UX)

Pada halaman sign up ini memiliki beberapa label, seperti label nama, email dan password. Label-label tersebut digunakan untuk pendaftaran sebuah akun untuk menggunakna aplikasi ini. Fungsi dari halaman sign up (daftar) pada sebuah aplikasi media sosial adalah memungkinkan pengguna baru untuk membuat akun di aplikasi media sosial. Pengguna dapat memberikan informasi pribadi seperti nama, alamat email, kata sandi. Setelah mendaftar, pengguna dapat mengakses ke fitur-fitur dasar aplikasi seperti membuat postingan, menghubungkan dengan teman, dan mengatur profil.



Gambar 2.8 Nama, Email, Password

Berikut ini adalah bagian yang dimana untuk mengisi name atau biasa disebut dengan username, alamat email, dan juga password. Penggunaan tulisan dan garis tersebut berfungsi sebagai penanda masing-masing labek, sehinggal pengguna dapat mudah mengerti dan menggunakan fungsi-fungsi dari setiap label tersebut. Penggunaan warna abu-abu tersebut memberikan warna yang lebih netral, agar cocok terhadap latar belakang dari halaman ini.



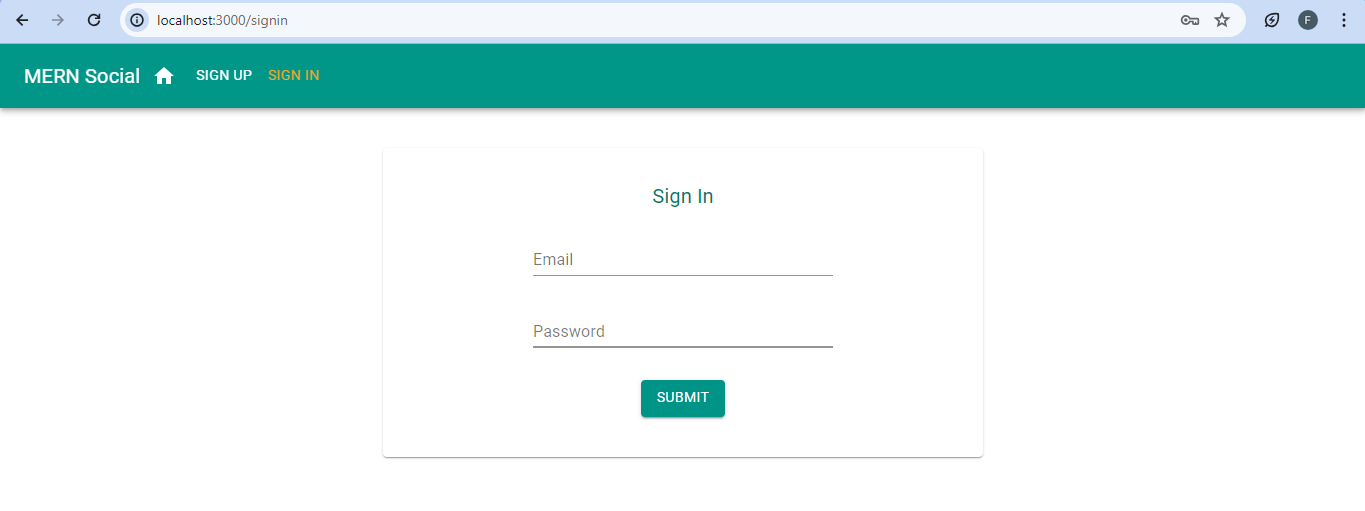
Gambar 2.9 Button Sign Up

Pada halaman ini tentu saja memerlukan tombol sign up yang digunakan untuk menjalankan sebuah aksi. Sehingga ketik menekan tombol sign up, akan menuju ke halaman sign up.



Gambar 2.10 Button Submit

Tombol submit ini menjalankan aksi, Ketika mengisi data kita dan memencet tombol ini maka data kita akan terdaftar.



Gambar 2.11 Halaman Sign In Sosial Meida (UI/UX)

Fungsi dari halaman Sign In (Masuk) pada sebuah aplikasi media sosial adalah

Memungkinkan pengguna yang sudah memiliki akun untuk masuk ke dalam aplikasi.

Pengguna akan diminta untuk memasukkan kredensial masuk, seperti alamat email dan kata sandi, untuk mengotentikasi identitas mereka. Setelah berhasil masuk, pengguna dapat mengakses semua fitur aplikasi yang tersedia untuk mereka, termasuk berinteraksi dengan teman, membaca konten, dan mengatur profil.



Gambar 2.12 Button Sign In

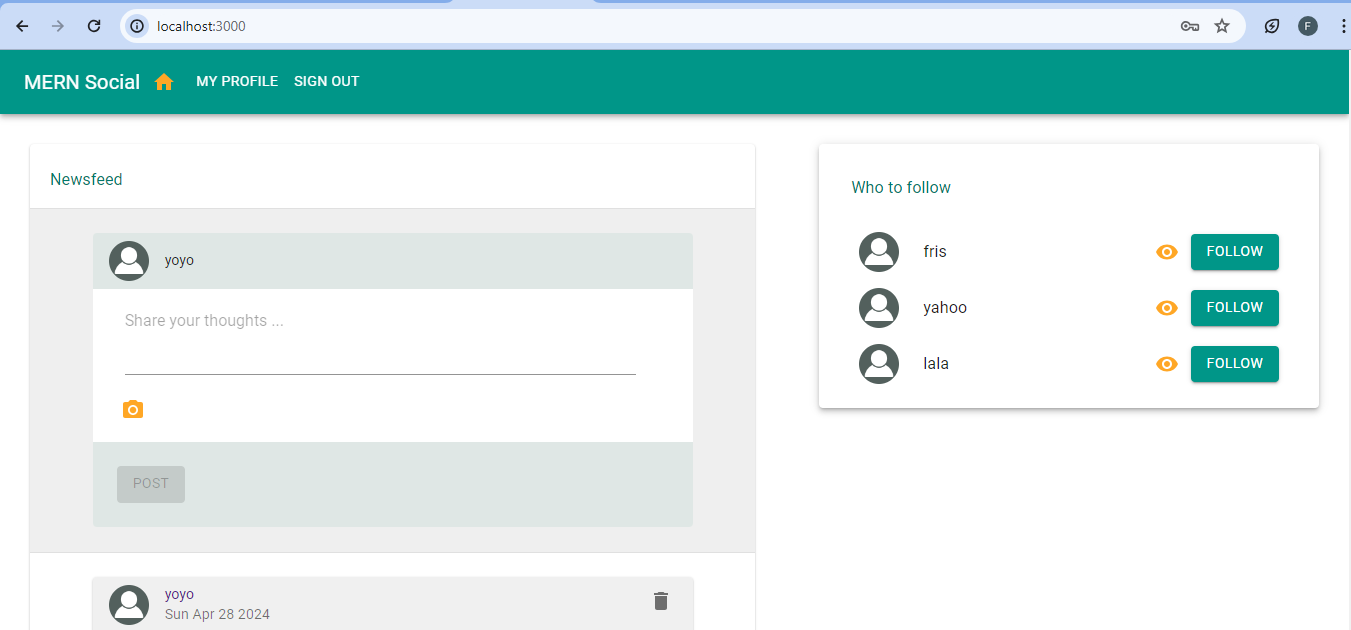
Tombol Sign In berfungsi untuk mengidentifikasi pengguna. Ketika pengguna mengklik tombol ini, mereka diarahkan ke halaman di mana pengguna harus memasukkan kredensial mereka, username dan password.



Gambar 2.13 Button Submit

Dan untuk tombol Submit ini berfungsi untuk mengirimkan formulir ke server. Tombol ini biasanya digunakan memungkinkan pengguna mengirimkan masukan mereka, seperti kredensial login, permintaan pencarian, atau detail pendaftaran

#### C. Halaman Beranda



Gambar 2.14 Halaman Beranda Sosial Media

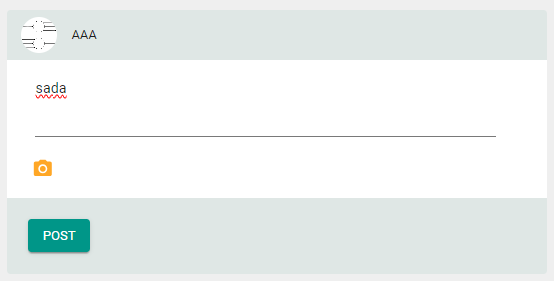
Fitur ini memungkinkan pengguna untuk membuat dan membagikan konten baru dengan pengguna lainnya. Pengguna dapat membuat posting teks atau gambar dan membagikannya di halaman utama atau di profil mereka.

Di dalam fungsi Post itu sendiri, terdapat beberapa keadaan (state) yang dikelola menggunakan hook useState. Ini termasuk status suka (like), jumlah suka (likes), dan komentar (comments). Nilai-nilai awalnya diatur menggunakan properti props.post yang diterima dari luar. File tersebut adalah sebuah komponen React yang disebut Post. Komponen ini digunakan untuk menampilkan postingan dalam aplikasi sosial media.



Gambar 2.15 Button Sign Out

Pada bagian Sign Out Button ini untuk mengakhiri sesi pengguna yang sedang aktif. Ketika pengguna mengklik tombol ini, sistem akan memutus koneksi antara pengguna dan akun mereka, menandakan bahwa sesi tersebut telah selesai. Dan hal Ini memastikan bahwa pengguna tidak lagi memiliki akses ke informasi dan fitur akun mereka sampai mereka masuk kembali.



Gambar 2.16 Button Post

Tombol ini untuk mengirimkan konten yang dibuat oleh pengguna ke server. Konten ini bisa berupa teks, gambar, video, atau kombinasi dari berbagai media. Setelah tombol "Post" diklik, data yang dimasukkan pengguna dikirimkan ke server untuk diproses dan disimpan.



Gambar 2.17 Ikon Camera

Pada tombol kamera ini berfungsi untuk mengaktifkan perangkat kamera. Ketika pengguna mengklik tombol ini, aplikasi atau perangkat akan membuka antarmuka kamera, siap untuk mengambil gambar atau video. Setelah kamera diaktifkan, tombol kamera berfungsi untuk mengambil gambar atau mulai merekam video. Mengklik tombol ini akan memicu rana kamera untuk menangkap gambar atau memulai proses perekaman.



Gambar 2.18 Ikon Trash

Ikon trash ini berguna untuk menghapus konten yang dipilih. Ini bisa berupa file, pesan, gambar, video, email, atau dokumen lainnya. Ketika pengguna mengklik tombol ini, item yang dipilih dipindahkan ke tempat sampah atau dihapus secara permanen tergantung pada pengaturan aplikasi atau sistem.





Gambar 2.19 Ikon Like

Fungsi utama ikon "Love" adalah untuk menyatakan bahwa pengguna menyukai atau menghargai konten tertentu, seperti postingan, komentar, foto, video, atau artikel. Dengan mengklik ikon "Love", pengguna menunjukkan apresiasi mereka terhadap konten tersebut.



Gambar 2.20 Comment

Bagian komentar memungkinkan pengguna untuk memberikan umpan balik langsung mengenai konten yang diposting, seperti artikel, video, blog, atau produk. Umpan balik ini bisa berupa apresiasi, kritik, saran, atau diskusi yang konstruktif yang membantu pembuat konten meningkatkan karya mereka.



Gambar 2.21 Delete Comment

Guna dari adanya fitur delete comment ini adalah untuk menghapus komentar yang tidak diinginkan. Tombol ini digunakan untuk menghapus komentar yang menyinggung, kasar, dan tidak relevan.



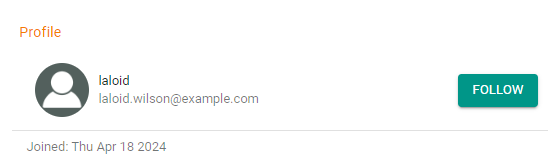
Gambar 2.22 Who to follow

Fitur ini memungkinkan pengguna untuk mengikuti akun atau profil pengguna lain. Dengan mengikuti, pengguna dapat melihat konten yang dibagikan oleh akun yang mereka ikuti di feed mereka dan tetap terhubung dengan orang-orang yang mereka minati. Dengan tulisan “Who to follow”, dengan ini menyampaikan fungsinya langsung kepada para pengguna. Dengan tulisan yang berwarna hijau sesuai dengan penggunaan warna dasar pada aplikasi ini.



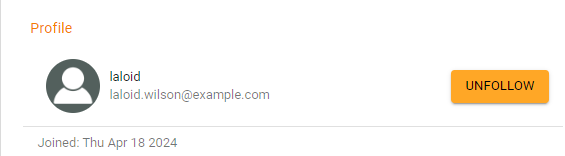
Gambar 2.23 Ikon See

Ikon ini digunakan untuk melihat gambar profil seseorang atau konten lainnya.



Gambar 2.24 Follow

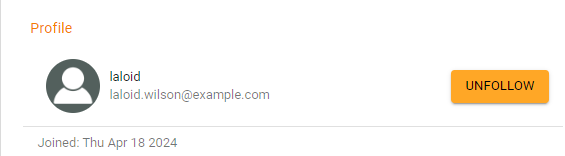
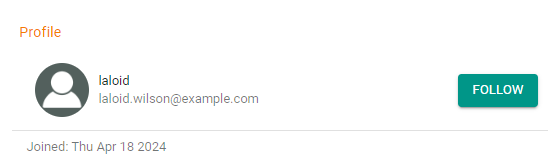
Tombol follow adalah fitur umum pada platform media sosial dan situs web lain yang memungkinkan pengguna untuk berlangganan atau mengikuti konten pengguna atau entitas lain. Saat pengguna mengikuti seseorang, mereka biasanya akan melihat postingan atau pembaruannya di umpan berita atau di beranda mereka. Hal ini dapat membantu pengguna tetap terhubung dengan orang-orang dan organisasi yang mereka minati, dan juga dapat membantu bisnis atau individu mempromosikan konten mereka kepada audiens yang lebih luas.



Gambar 2.25 Unfollow

Apabila pengguna telah mengklik tombol follow makanya akan berubah seperti yang ada pada Gambar 2.24.

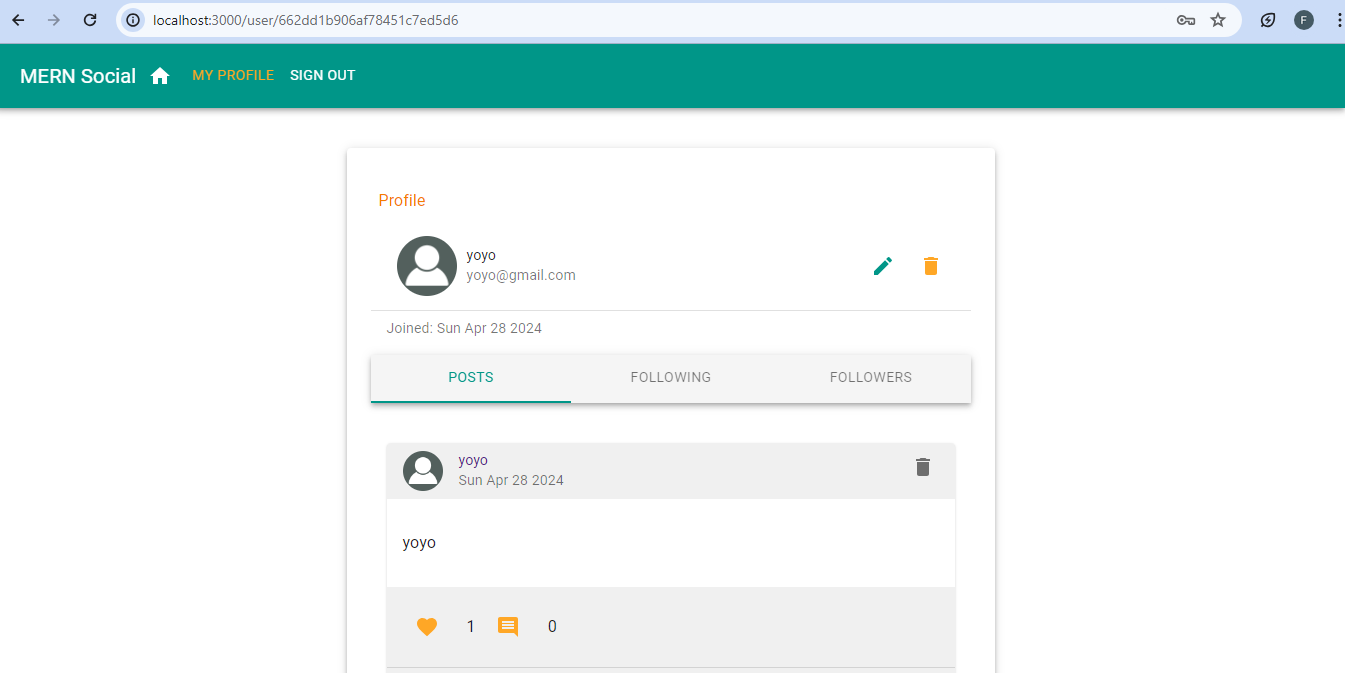
Note: Profile pengguna lainnya ini muncul ketika pengguna klik icon mata



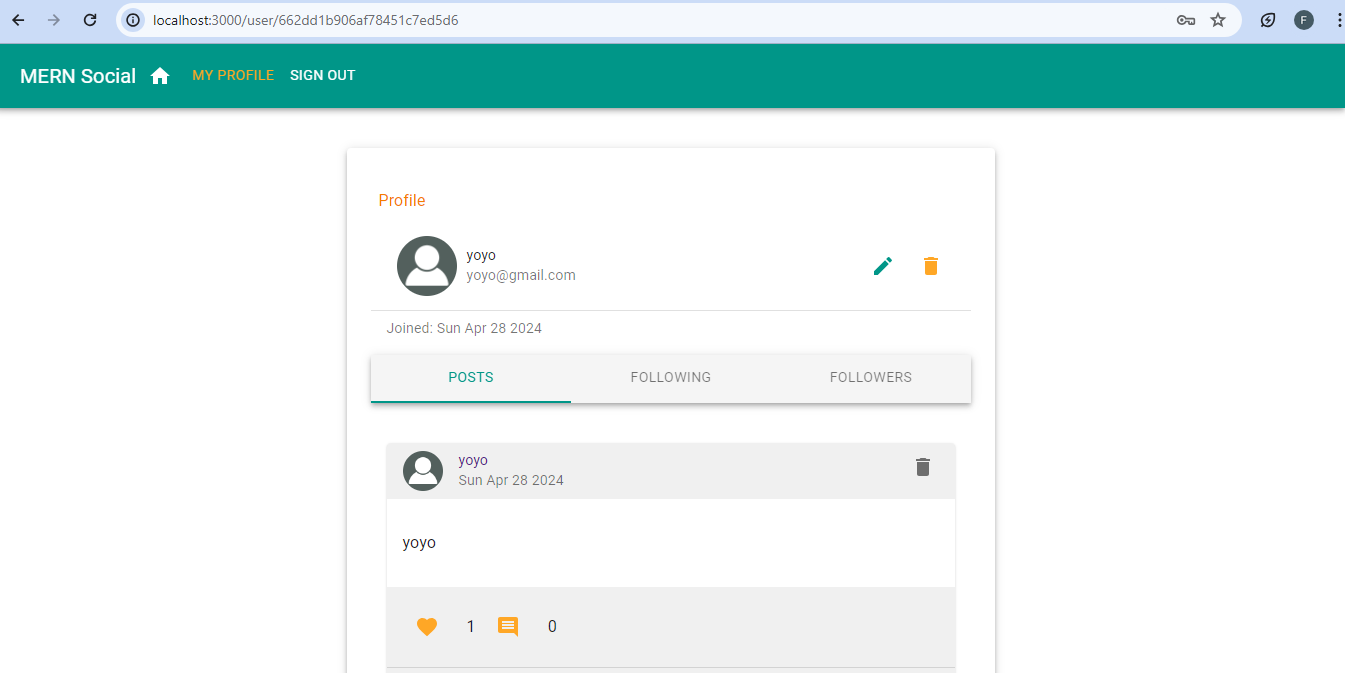
Gambar 2.26 Follow & Unfollow Button

#### D. Halaman My Profile

Fitur ini memungkinkan pengguna untuk mengakses dan mengelola profil mereka. Mereka dapat melihat informasi pribadi mereka, mengubah nama akun, menghapus akun, dan melihat pengikut dan mengikuti serta postingan dari akun kita.

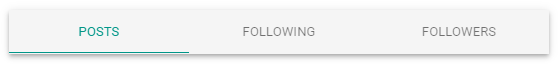


Gambar 2.27 Halaman My Profile



Gambar 2.28 Ikon Edit & Delete

Tombol tanda pena menandakan bahwa tombol ini untuk mengedit, yang letaknya di posisi samping profile ini dapat memudahkan pengguna untuk mengerti fungsinya. Dan tulisan “Joined Sun Apr 28 2024”, memberikan informasi kapan pengguna membuat akun ini. Ikon tong sampah ini memberikan makna menghapus, dengan ini pengguna dapat langsung mengetahui fungsi dari ikon ini yang letaknya dekat dengan ikon pena dan profil. Pemilihan warna oren agar sesuai dengan warna-warna ikon seperti like yang sudah digunakan. Pemilihan warna yang berbeda dengan ikon edit, supaya pengguna tidak bingung dan dapat langsung melihat perbedaannya dikarena posisi mereka yang bersebelahan.



Gambar 2.29 Tab Posts, Following, Followers

Pada bagian post ini akan menampilkan postingan atau konten apa saja yang telah dibagikan oleh pengguna. pada bagian following ini akan menampilkan beberapa akun yang diikuti oleh pengguna. Dan untuk followers akan menampilkan akun-akun yang menggikuti sang pengguna. Pada dasarnya, following adalah tentang secara aktif mencari konten dari orang lain, sedangkan followers adalah tentang audiens yang datang kepada Anda. Baik following dan followers memainkan peran penting dalam engagement media sosial dan visibilitas konten.

## 3.1.2.2 Coding

### A. Halaman Utama



Gambar 2.30 Ikon Home

Source code tautan yang membawa pengguna ke tampilan Home:

<Link to="/">

        <IconButton aria-label="Home" style={isActive(history, "/")}>

          <HomeIcon/>

        </IconButton>

      </Link>

1. Link to

<Link to="/">

*<Link to="/">* adalah komponen *Link* dari *React Router* yang menghubungkan tautan ini dengan halaman beranda *("/").* Ketika tautan ini diklik, navigasi akan dialihkan ke halaman beranda.

1. *Home Button*

        <IconButton aria-label="Home" style={isActive(history, "/")}>

          <HomeIcon/>

        </IconButton>

      </Link>

* + *<IconButton aria-label="Home" style={isActive(history, "/")}>* adalah komponen *IconButton* dari *material-ui* yang menampilkan ikon rumah. Properti *aria-label* memberikan label aksesibilitas untuk ikon tersebut. Properti *style* digunakan untuk menentukan apakah tautan ini aktif atau tidak dengan menggunakan fungsi *isActive(history, "/").* Fungsi ini kemungkinan besar memeriksa apakah halaman saat ini sesuai dengan tautan yang diberikan.
  + *<HomeIcon/>* adalah komponen ikon rumah yang akan ditampilkan di dalam tombol.

Source code yang menjalankan tampilan Home yang akan dilihat pengguna sebelum signin:

export default function Home({ history }) {

  const classes = useStyles();

  const [defaultPage, setDefaultPage] = useState(false);

  useEffect(() => {

    setDefaultPage(auth.isAuthenticated());

    const unlisten = history.listen(() => {

      setDefaultPage(auth.isAuthenticated());

    });

    return () => {

      unlisten();

    };

  }, []);

  return (

    <div className={classes.root}>

      {!defaultPage && (

        <Grid container spacing={8}>

          <Grid item xs={12}>

            <Card className={classes.card}>

              <Typography variant="h6" className={classes.title}>

                Home Page

              </Typography>

              <CardMedia

                className={classes.media}

                image={socialMediaImg}

                title="Social Media Logo"

              />

              <Typography

                variant="body2"

                component="p"

                className={classes.credit}

                color="textSecondary"

              >

                Photo by{" "}

                <a

                  href="https://unsplash.com/@boudewijn\_huysmans"

                  target="\_blank"

                  rel="noopener noreferrer"

                >

                  Boudewijn Huysmans

                </a>{" "}

                on Unsplash

              </Typography>

              <CardContent>

                <Typography type="body1" component="p">

                  Welcome to the MERN Social home page.

                </Typography>

              </CardContent>

            </Card>

          </Grid>

        </Grid>

      )}

1. *Home function*

export default function Home({ history }) {

Mendefinisikan dan mengekspor secara default sebuah fungsi komponen *Home* yang menerima *history* sebagai properti (*prop*)

1. *useStyles & useState*

  const classes = useStyles();

  const [defaultPage, setDefaultPage] = useState(false);

*const classes = useStyles();* memanggil fungsi *useStyles* yang mengembalikan objek berisi kelas-kelas *CSS* yang akan digunakan untuk styling komponen ini. *const [defaultPage, setDefaultPage] = useState(false);* menggunakan hook *useState* untuk membuat state bernama *defaultPage* yang awalnya bernilai *false*. *setDefaultPage* adalah fungsi untuk mengubah nilai *defaultPage.*

1. *useEffect*

  useEffect(() => {

    setDefaultPage(auth.isAuthenticated());

    const unlisten = history.listen(() => {

      setDefaultPage(auth.isAuthenticated());

    });

    return () => {

      unlisten();

    };

  }, []);

* *useEffect(() => { ... }, []);* adalah *hoosk* *useEffect* yang menjalankan efek samping saat komponen dipasang (*mounted*) dan dibersihkan saat komponen dilepas (*unmounted*).
* *setDefaultPage(auth.isAuthenticated());* memanggil fungsi *auth.isAuthenticated()* untuk memeriksa apakah pengguna sudah login dan mengatur defaultPage sesuai hasilnya.
* *const unlisten = history.listen(() => { ... });* mencari perubahan pada objek *history* (misalnya saat pengguna berpindah halaman) dan mengatur *defaultPage* lagi.
* *return () => { unlisten(); };* membersihkan listener saat komponen dilepas (*unmounted*).

1. *Render* Tampilan

  return (

    <div className={classes.root}>

      {!defaultPage && (

        <Grid container spacing={8}>

          <Grid item xs={12}>

            <Card className={classes.card}>

              <Typography variant="h6" className={classes.title}>

                Home Page

              </Typography>

              <CardMedia

                className={classes.media}

                image={socialMediaImg}

                title="Social Media Logo"

              />

              <Typography

                variant="body2"

                component="p"

                className={classes.credit}

                color="textSecondary"

              >

                Photo by{" "}

                <a

                  href="https://unsplash.com/@boudewijn\_huysmans"

                  target="\_blank"

                  rel="noopener noreferrer"

                >

                  Boudewijn Huysmans

                </a>{" "}

                on Unsplash

              </Typography>

              <CardContent>

                <Typography type="body1" component="p">

                  Welcome to the MERN Social home page.

                </Typography>

              </CardContent>

            </Card>

          </Grid>

        </Grid>

      )}

* *return ( ... );* adalah bagian yang mengembalikan *JSX* untuk merender tampilan komponen.
* *<div className={classes.root}>:* Elemen *div* dengan kelas *root*.
* *{!defaultPage && ( ... )}:* Jika *defaultPage* adalah false, maka konten di dalam kurung kurawal akan dirender.
* *<Grid container spacing={8}>:* Grid dari *Material-UI* dengan jarak antar item sebesar 8.
* *<Card className={classes.card}>:* Komponen *Card* dari *Material-UI* untuk menampilkan konten dalam bentuk kartu.
* *<Typography variant="h6" className={classes.title}>Home Page</Typography>*: Menampilkan teks "*Home Page*" dengan gaya *Typography* dari *Material-UI*.
* <*CardMedia className={classes.media} image={socialMediaImg} title="Social Media Logo" />:* Menampilkan gambar dengan gaya *CardMedia*.
* *<Typography variant="body2" component="p" className={classes.credit} color="textSecondary">Photo by* *...</Typography>:* Menampilkan kredit untuk foto dengan gaya *Typography*.
* *<CardContent>...<Typography type="body1" component="p">Welcome to the MERN Social home page.</Typography>...</CardContent>:* Menampilkan konten kartu dengan teks selamat datang.

### B. Sign Up (Daftar)

Source code yang membawa user ke tampilan Signup ketika diklik:

 <Link to="/signup">

            <Button style={isActive(history, "/signup")}>Sign up

            </Button>

          </Link>

Kode ini membuat tautan (*link*) yang menuju ke halaman "*Sign Up*". Saat pengguna mengklik tombol "*Sign up*", mereka akan diarahkan ke halaman *"/signup"*.

Source code yang menjalankan tampilan atau display signup yang dilihat oleh pengguna:

export default function Signup (){

  const classes = useStyles()

  const [values, setValues] = useState({

    name: '',

    password: '',

    email: '',

    open: false,

    error: ''

  })

  const handleChange = name => event => {

    setValues({ ...values, [name]: event.target.value })

  }

  const clickSubmit = () => {

    const user = {

      name: values.name || undefined,

      email: values.email || undefined,

      password: values.password || undefined

    }

    create(user).then((data) => {

      if (data.error) {

        setValues({ ...values, error: data.error})

      } else {

        setValues({ ...values, error: '', open: true})

      }

    })

  }

    return (<div>

      <Card className={classes.card}>

        <CardContent>

          <Typography variant="h6" className={classes.title}>

            Sign Up

          </Typography>

          <TextField id="name" label="Name" className={classes.textField} value={values.name} onChange={handleChange('name')} margin="normal"/><br/>

          <TextField id="email" type="email" label="Email" className={classes.textField} value={values.email} onChange={handleChange('email')} margin="normal"/><br/>

          <TextField id="password" type="password" label="Password" className={classes.textField} value={values.password} onChange={handleChange('password')} margin="normal"/>

          <br/> {

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}</Typography>)

          }

        </CardContent>

* + 1. *Signup function*

export default function Signup (){

  const classes = useStyles()

  const [values, setValues] = useState({

    name: '',

    password: '',

    email: '',

    open: false,

    error: ''

  })

*useState* digunakan untuk menyimpan dan memperbarui nilai form input seperti *name*, *email*, dan *password*.

  const handleChange = name => event => {

    setValues({ ...values, [name]: event.target.value })

  }

*handleChange* adalah fungsi yang memperbarui nilai state ketika pengguna mengetik di input.

  const clickSubmit = () => {

    const user = {

      name: values.name || undefined,

      email: values.email || undefined,

      password: values.password || undefined

    }

    create(user).then((data) => {

      if (data.error) {

        setValues({ ...values, error: data.error})

      } else {

        setValues({ ...values, error: '', open: true})

      }

    })

  }

*clickSubmit* adalah fungsi yang dipanggil ketika tombol "*Submit*" diklik. *clickSubmit* akan mengirim data ke server untuk membuat akun baru.

    return (<div>

      <Card className={classes.card}>

        <CardContent>

          <Typography variant="h6" className={classes.title}>

            Sign Up

          </Typography>

          <TextField id="name" label="Name" className={classes.textField} value={values.name} onChange={handleChange('name')} margin="normal"/><br/>

          <TextField id="email" type="email" label="Email" className={classes.textField} value={values.email} onChange={handleChange('email')} margin="normal"/><br/>

          <TextField id="password" type="password" label="Password" className={classes.textField} value={values.password} onChange={handleChange('password')} margin="normal"/>

Kode ini adalah bagian dari komponen *React* yang menggunakan *Material-UI* untuk membangun sebuah form pendaftaran *(Sign Up).*

* *<Card>* adalah komponen dari *Material-UI* yang berfungsi seperti kotak atau wadah untuk konten.
* *className={classes.card}* memberikan kelas CSS yang dinamakan card dari objek classes untuk mengatur gaya tampilan kartu ini.
* *<CardContent>:* Komponen ini digunakan untuk menampung konten di dalam kartu.
* *<Typography>* adalah komponen untuk mengatur tampilan teks.
* *variant="h6"* berarti teks ini akan ditampilkan dengan gaya heading 6.
* *className={classes.title}* memberikan kelas *CSS* yang dinamakan title untuk mengatur gaya teks ini.
* Konten di dalamnya adalah "*Sign Up*", yang berarti teks "*Sign Up*" akan muncul di layar.
* *<TextField>* adalah komponen input dari *Material-UI* untuk memasukkan teks.
* *id, label, type,* dan *className* adalah atribut untuk mengatur properti dan gaya masing-masing input.
* *value={values.name}, value={values.email}, value={values.password}* menghubungkan nilai input dengan state (status) yang bernama values (misalnya, values.name untuk input name).
* *onChange={handleChange('name')}, onChange={handleChange('email')}, onChange={handleChange('password')}* adalah fungsi yang dipanggil saat nilai input berubah, untuk memperbarui state.

          <br/> {

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}</Typography>)

          }

        </CardContent>

Logika kondisional untuk menampilkan pesan error jika ada.

* *values.error* adalah nilai dari *state* yang menandakan apakah ada *error*.
* Jika *values.error* benar (ada *error*), maka akan menampilkan:
* *<Typography>* untuk teks pesan *error*.
* *<Icon>* dengan ikon *error*.
* *values.error* untuk menampilkan pesan *error* itu sendiri.

Source code untuk button *Submit* pada form *Signup*, source code ini juga memberi kotak dialog kepada pengguna apabila akun telah sukses didaftarkan:

        <CardActions>

          <Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>

        </CardActions>

      </Card>

      <Dialog open={values.open} disableBackdropClick={true}>

        <DialogTitle>New Account</DialogTitle>

        <DialogContent>

          <DialogContentText>

            New account successfully created.

          </DialogContentText>

        </DialogContent>

* + - * *CardActions*

        <CardActions>

          <Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>

        </CardActions>

      </Card>

*<CardActions>* adalah komponen dari Material-UI yang digunakan untuk menempatkan aksi atau tombol yang berhubungan dengan konten kartu. Di dalamnya, ada:

* komponen *<Button>: color="primary"* memberikan warna utama (biasanya biru) pada tombol. *variant="contained"* membuat tombol tampak *solid* dan terangkat.
* *onClick={clickSubmit}* menetapkan fungsi *clickSubmit* yang akan dipanggil ketika tombol diklik.
* *className={classes.submit}* memberikan kelas *CSS* yang dinamakan submit untuk mengatur gaya tombol.
* Tulisan di dalam tombol adalah "*Submit*".
  + - * *Dialog*

      <Dialog open={values.open} disableBackdropClick={true}>

        <DialogTitle>New Account</DialogTitle>

        <DialogContent>

          <DialogContentText>

            New account successfully created.

          </DialogContentText>

        </DialogContent>

*<Dialog>* adalah komponen dari *Material-UI* untuk membuat *dialog* atau *pop-up.*

* *open={values.open}* menentukan apakah dialog akan terbuka atau tidak, berdasarkan nilai values.open.
* *disableBackdropClick={true}* mencegah dialog ditutup dengan mengklik area di luar dialog.

*<DialogTitle>* memberikan judul untuk *dialog*. Dalam kode ini, judulnya adalah "*New Account*".

*<DialogContent>* adalah konten utama dari dialog. Di dalamnya ada *<DialogContentText>* yang digunakan untuk teks konten dialog. Teksnya adalah "*New account successfully created*."

Source code ini membawa pengguna ke tampilan Sign in agar pengguna bisa masuk ke akun baru mereka:

        <DialogActions>

          <Link to="/signin">

            <Button color="primary" autoFocus="autoFocus" variant="contained">

              Sign In

            </Button>

          </Link>

        </DialogActions>

      </Dialog>

    </div>)

### C. Sign In (Masuk)

Yang dijalankan setelah pengguna sign in:

      {defaultPage && (

        <Grid container spacing={8}>

          <Grid item xs={8} sm={7}>

            <Newsfeed />

          </Grid>

          <Grid item xs={6} sm={5}>

            <FindPeople />

          </Grid>

        </Grid>

      )}

    </div>

  );

Source code signin button yang membawa pengguna ke tampilan Sign In:

<Link to="/signin">

            <Button style={isActive(history, "/signin")}>Sign In

            </Button>

          </Link>

Kode ini membuat tautan yang menuju ke halaman "*Sign In*". Saat pengguna mengklik tombol "*Sign in*", mereka akan diarahkan ke halaman *"/signin".*

Metode fetch signin yang akan mengirim request POST request untuk verifikasi kepada bagian backend, terletak pada file api-auth.js :

const signin = async (user) => {

  try {

    let response = await fetch('/auth/signin/', {

      method: 'POST',

      headers: {

        'Accept': 'application/json',

        'Content-Type': 'application/json'

      },

      credentials: 'include',

      body: JSON.stringify(user)

    })

    return await response.json()

  } catch(err) {

    console.log(err)

  }

}

export {

  signin,

signout

}

Fungsi signin mengirim permintaan *POST* ke *server* dengan data pengguna (*email* dan *password*) untuk masuk. Jika berhasil, *server* akan mengembalikan respons *JSON*.

* + *Const signin*

const signin = async (user) => {

Mendefinisikan fungsi *signin* sebagai fungsi *asinkron (async).* Fungsi ini menerima satu argumen, *user*, yang merupakan objek berisi informasi pengguna seperti *email* dan *password.*

* + *Try & Catch*

  try {

    let response = await fetch('/auth/signin/', {

      method: 'POST',

      headers: {

        'Accept': 'application/json',

        'Content-Type': 'application/json'

      },

      credentials: 'include',

      body: JSON.stringify(user)

    })

    return await response.json()

  } catch(err) {

    console.log(err)

  }

}

Blok *try...catch* digunakan untuk menangani kesalahan yang mungkin terjadi selama eksekusi fungsi asinkron. Jika ada kesalahan, pesan kesalahan akan ditangkap di blok *catch* dan dicetak ke konsol.

* *fetch* adalah fungsi bawaan *JavaScript* yang digunakan untuk membuat permintaan *HTTP*. Dalam kasus ini, permintaan *POST* dikirim ke *endpoint /auth/signin/.*
* *await* digunakan untuk menunggu sampai permintaan selesai dan respons diterima.
* *method: 'POST'* menentukan metode *HTTP* yang digunakan, yaitu *POST*.
* *headers: { ... }* menentukan tipe konten yang dikirim dalam permintaan. Dalam kasus ini, data dikirim dalam format *JSON*:
* *'Accept': 'application/json'* memberitahu server bahwa klien mengharapkan respons dalam format *JSON*.
* *'Content-Type': 'application/json'* memberitahu server bahwa data yang dikirim adalah dalam format *JSON*.
* *credentials: 'include'* menentukan apakah permintaan harus menyertakan kredensial seperti *cookie*. *'include*' memastikan bahwa *cookie* juga dikirim bersama permintaan.
* *body: JSON.stringify(user)* berisi data yang dikirim dalam permintaan. Data pengguna (*user*) diubah menjadi string *JSON* menggunakan *JSON.stringify(user).*
* *return await response.json()* menunggu respons dari server dan mengembalikan respons dalam format *JSON*. await digunakan untuk menunggu hingga respons diproses menjadi JSON.
* *catch (err) { console.log(err); }* menangkap kesalahan yang terjadi dalam blok *try* dan mencetak pesan kesalahan ke konsol.
  + *Export*

export {

  signin,

signout

}

*export { ... }* mengekspor signin dan signout dari modul ini sehingga dapat digunakan di bagian lain dari aplikasi. Ekspor ini memungkinkan fungsi *signin* dan *signout* diimpor dan digunakan di komponen lain atau file lain dalam proyek.

Source code yang menjalan form Signin:

export default function Signin(props) {

  const classes = useStyles();

  const [values, setValues] = useState({

    email: "",

    password: "",

    error: "",

    redirectToReferrer: false,

  });

  const clickSubmit = () => {

    const user = {

      email: values.email || undefined,

      password: values.password || undefined,

    };

    signin(user).then((data) => {

      if (data.error) {

        setValues({ ...values, error: data.error });

      } else {

        auth.authenticate(data, () => {

          setValues({ ...values, error: "", redirectToReferrer: true });

        });

      }

    });

  };

  const handleChange = (name) => (event) => {

    setValues({ ...values, [name]: event.target.value });

  };

  const { from } = props.location.state || {

    from: {

      pathname: "/",

    },

  };

  const { redirectToReferrer } = values;

  if (redirectToReferrer) {

    return <Redirect to={from} />;

  }

  return (

    <Card className={classes.card}>

      <CardContent>

        <Typography variant="h6" className={classes.title}>

          Sign In

        </Typography>

        <TextField

          id="email"

          type="email"

          label="Email"

          className={classes.textField}

          value={values.email}

          onChange={handleChange("email")}

          margin="normal"

        />

        <br />

        <TextField

          id="password"

          type="password"

          label="Password"

          className={classes.textField}

          value={values.password}

          onChange={handleChange("password")}

          margin="normal"

        />

        <br />{" "}

        {values.error && (

          <Typography component="p" color="error">

            <Icon color="error" className={classes.error}>

              error

            </Icon>

            {values.error}

          </Typography>

        )}

      </CardContent>

* 1. *Signin function*

export default function Signin(props) {

  const classes = useStyles();

  const [values, setValues] = useState({

    email: "",

    password: "",

    error: "",

    redirectToReferrer: false,

  });

*useStyles* digunakan untuk mengambil kelas CSS yang didefinisikan untuk styling komponen. *useState* digunakan untuk mendefinisikan *state* lokal komponen, yaitu *email, password, error*, dan *redirectToReferrer*.

* 1. *clickSubmit*

  const clickSubmit = () => {

    const user = {

      email: values.email || undefined,

      password: values.password || undefined,

    };

    signin(user).then((data) => {

      if (data.error) {

        setValues({ ...values, error: data.error });

      } else {

        auth.authenticate(data, () => {

          setValues({ ...values, error: "", redirectToReferrer: true });

        });

      }

    });

  };

Fungsi ini dipanggil ketika tombol "Submit" diklik.

* Membuat objek user yang berisi *email* dan *password* dari state.
* Memanggil fungsi signin yang telah dijelaskan sebelumnya untuk mengirim data ke *server*.
* Jika ada kesalahan (*data.error*), memperbarui state error. Jika berhasil, memanggil *auth.authenticate* untuk mengautentikasi pengguna dan mengatur *redirectToReferrer* menjadi true.
  1. *handleChange*

  const handleChange = (name) => (event) => {

    setValues({ ...values, [name]: event.target.value });

  };

Fungsi ini digunakan untuk memperbarui state saat pengguna mengetik di input form dengan mengambil *name* dari parameter dan *event* dari *input*, kemudian memperbarui nilai *state* yang sesuai.

* 1. *Redirect*

  const { from } = props.location.state || {

    from: {

      pathname: "/",

    },

  };

  const { redirectToReferrer } = values;

  if (redirectToReferrer) {

    return <Redirect to={from} />;

  }

*const { from } = props.location.state || { from: { pathname: "/" } }* digunakan untuk menentukan halaman mana yang akan dituju setelah pengguna berhasil sign in. Jika *redirectToReferrer* adalah true, maka pengguna akan diarahkan ke halaman yang ditentukan oleh from.

* 1. Tampilkan *form*

  return (

    <Card className={classes.card}>

      <CardContent>

        <Typography variant="h6" className={classes.title}>

          Sign In

        </Typography>

        <TextField

          id="email"

          type="email"

          label="Email"

          className={classes.textField}

          value={values.email}

          onChange={handleChange("email")}

          margin="normal"

        />

        <br />

        <TextField

          id="password"

          type="password"

          label="Password"

          className={classes.textField}

          value={values.password}

          onChange={handleChange("password")}

          margin="normal"

        />

        <br />{" "}

        {values.error && (

          <Typography component="p" color="error">

            <Icon color="error" className={classes.error}>

              error

            </Icon>

            {values.error}

          </Typography>

        )}

      </CardContent>

Menggunakan komponen *Material-UI* seperti *Card, CardContent, Typography, TextField,* dan *Button* untuk membuat tampilan form yang rapi. Jika ada kesalahan (*values.error*), menampilkan pesan kesalahan di bawah input form.

Button Submit  yang pengguna gunakan untuk Signin:

      <CardActions>

<Button

          color="primary"

          variant="contained"

          onClick={clickSubmit}

          className={classes.submit}

        >

          Submit

        </Button>

      </CardActions>

    </Card>

  );

}

* + 1. *CardActions*

      <CardActions>

Komponen dari *Material-UI* yang digunakan untuk mengatur tata letak tombol-tombol atau aksi lainnya di dalam sebuah kartu (*card*).

* + 1. *Button*

<Button

          color="primary"

          variant="contained"

          onClick={clickSubmit}

          className={classes.submit}

        >

          Submit

        </Button>

      </CardActions>

Komponen tombol dari Material-UI. Submit adalah teks yang akan ditampilkan di dalam tombol. Tombol ini memiliki beberapa properti (attributes) yang diatur:

* *color="primary":* Mengatur warna tombol menjadi warna utama.
* *variant="contained":* Menjadikan tombol terlihat solid dan menonjol (bukan hanya teks).
* *onClick={clickSubmit}:* Menetapkan fungsi clickSubmit yang akan dipanggil saat tombol diklik.
* *className={classes.submit}:* Menggunakan kelas CSS untuk mengatur gaya (*styling*) tombol, yang didefinisikan di *classes.submit*.

Fungsi signin digunakan untuk mengirim permintaan HTTP POST ke endpoint /auth/signin/ pada server. Permintaan ini berisi data pengguna yang ingin masuk. Setelah menerima respons dari server, fungsi ini mengembalikan data JSON yang berisi informasi tentang pengguna yang berhasil masuk. Jika terjadi kesalahan selama proses, fungsi ini akan menangkap kesalahan dan mencetaknya ke konsol.

### D. Sign Out (Keluar)

Source code yang membiarkan pengguna keluar dari akun mereka:

<Button color="inherit" onClick={() => {

              auth.clearJWT(() => history.push('/'))

            }}>Sign out</Button>

        </span>)

Koding ini menampilkan button dengan teks "*Sign out*" dan menangani *Click Event "onClick".* Ketika button diketik, *auth.clearJWT* dipanggil untuk menghapus *JWT* (logout pengguna). “*history.push('/')*” digunakan untuk mengarahkan pengguna ke beranda*.*

Metode fetch signout digunakan untuk membuat request GET untuk memanggil API signout, source code ini terletak pada api-auth.js:

const signout = async () => {

  try {

    let response = await fetch('/auth/signout/', { method: 'GET' })

    return await response.json()

  } catch(err) {

    console.log(err)

  }

}

export {

  signin,

  signout

}

* 1. *Fetch signout*

const signout = async () => {

Baris ini mendefinisikan fungsi yang disebut signout yang tidak memerlukan argumen apa pun. Kata kunci *async* memungkinkan penggunaan “*await*” dalam fungsi ini, yang digunakan untuk menangani operasi *asynchronous* (hal-hal yang memerlukan waktu, seperti mengambil data dari server).

  try {

Mulai dengan blok *try*, yang berisi kode yang akan menangkap kesalahan.

    let response = await fetch('/auth/signout/', { method: 'GET' })

Baris ini mengirimkan permintaan *GET* ke server di *endpoint* */auth/signout/* untuk mengeluarkan pengguna. “*await*” memastikan bahwa kode menunggu di sini hingga server merespons.

    return await response.json()

Setelah server merespons, baris ini mengubah respons ke format *JSON* dan mengembalikannya. Sekali lagi, “*await*” membuat kode menunggu hingga respons sepenuhnya dikonversi.

  } catch(err) {

Jika ada kesalahan yang terjadi pada blok try, blok *catch* akan menangkap kesalahan tersebut.

    console.log(err)

  }

}

Baris ini mencetak kesalahan ke konsol, sehingga dapat *error* yang terjadi.

* 1. *Export*

export {

  signin,

  signout

}

Baris ini mengekspor fungsi *signin* dan *signout* sehingga dapat digunakan di file lain, seperti file *signout* ini

Fungsi ini memanfaatkan metode fetch untuk berkomunikasi dengan server secara asynchronous (non-blocking) dan menggunakan async/await untuk menangani respons dari server secara efisien. Selain itu, penggunaan credentials: 'include' dalam permintaan signin memastikan bahwa kredensial pengguna disertakan dalam setiap permintaan, yang penting untuk menjaga sesi masuk pengguna di aplikasi.

### E. My Profile

Source code yang membawa pengguna ke tampilan My Profile ketika pengguna klik menu My Profile:

<Link to={"/user/" + auth.isAuthenticated().user.\_id}>

            <Button style={isActive(history, "/user/" + auth.isAuthenticated().user.\_id)}>My Profile</Button>

          </Link>

1. *Link* *to*

<Link to={"/user/" + auth.isAuthenticated().user.\_id}>

*<Link>* adalah komponen dari *react-router-dom*, sebuah *library* untuk menangani navigasi di aplikasi *React*. Properti *to* menentukan ke mana tautan akan mengarahkan pengguna. Di sini, *URL*nya dibuat secara dinamis dengan menggabungkan string *"/user/"* dan ID pengguna yang diautentikasi.

1. *Button*

            <Button style={isActive(history, "/user/" + auth.isAuthenticated().user.\_id)}>My Profile</Button>

          </Link>

*<Button>* adalah komponen tombol. Di dalam *Link*, tombol ini akan terlihat seperti tombol biasa, tetapi bertindak sebagai tautan.

* Properti *style* digunakan untuk mengatur gaya tombol.
* Fungsi *isActive* dipanggil untuk menentukan gaya berdasarkan apakah *URL* saat ini cocok dengan *URL* tujuan.

Source code yang menjalankan tampilan My Profile yang pengguna melihat:

export default function Profile({ match }) {

  const classes = useStyles()

  const [values, setValues] = useState({

    user: {following:[], followers:[]},

    redirectToSignin: false,

    following: false

  })

  const [posts, setPosts] = useState([])

  const jwt = auth.isAuthenticated()

  useEffect(() => {

    const abortController = new AbortController()

    const signal = abortController.signal

    read({

      userId: match.params.userId

    }, {t: jwt.token}, signal).then((data) => {

      if (data && data.error) {

        setValues({...values, redirectToSignin: true})

      } else {

        let following = checkFollow(data)

        setValues({...values, user: data, following: following})

        loadPosts(data.\_id)

      }

    })

    return function cleanup(){

      abortController.abort()

    }

  }, [match.params.userId])

  const checkFollow = (user) => {

    const match = user.followers.some((follower)=> {

      return follower.\_id == jwt.user.\_id

    })

    return match

  }

  const clickFollowButton = (callApi) => {

    callApi({

      userId: jwt.user.\_id

    }, {

      t: jwt.token

    }, values.user.\_id).then((data) => {

      if (data.error) {

        setValues({...values, error: data.error})

      } else {

        setValues({...values, user: data, following: !values.following})

      }

    })

  }

  const loadPosts = (user) => {

    listByUser({

      userId: user

    }, {

      t: jwt.token

    }).then((data) => {

      if (data.error) {

        console.log(data.error)

      } else {

        setPosts(data)

      }

    })

  }

  const removePost = (post) => {

    const updatedPosts = posts

    const index = updatedPosts.indexOf(post)

    updatedPosts.splice(index, 1)

    setPosts(updatedPosts)

  }

    const photoUrl = values.user.\_id

              ? `/api/users/photo/${values.user.\_id}?${new Date().getTime()}`

              : '/api/users/defaultphoto'

    if (values.redirectToSignin) {

      return <Redirect to='/signin'/>

    }

    return (

      <Paper className={classes.root} elevation={4}>

        <Typography variant="h6" className={classes.title}>

          Profile

        </Typography>

        <List dense>

          <ListItem>

            <ListItemAvatar>

              <Avatar src={photoUrl} className={classes.bigAvatar}/>

            </ListItemAvatar>

            <ListItemText primary={values.user.name} secondary={values.user.email}/> {

             auth.isAuthenticated().user && auth.isAuthenticated().user.\_id == values.user.\_id

             ? (<ListItemSecondaryAction>

                  <Link to={"/user/edit/" + values.user.\_id}>

                    <IconButton aria-label="Edit" color="primary">

                      <Edit/>

                    </IconButton>

                  </Link>

                  <DeleteUser userId={values.user.\_id}/>

                </ListItemSecondaryAction>)

            : (<FollowProfileButton following={values.following} onButtonClick={clickFollowButton}/>)

            }

          </ListItem>

          <Divider/>

          <ListItem>

            <ListItemText primary={values.user.about} secondary={"Joined: " + (

              new Date(values.user.created)).toDateString()}/>

          </ListItem>

        </List>

        <ProfileTabs user={values.user} posts={posts} removePostUpdate={removePost}/>

      </Paper>

    )

}

1. *Profile function*

export default function Profile({ match }) {

* *export default* menandakan bahwa ini adalah komponen utama yang diekspor dari file ini. *Profile* adalah nama komponen.
* *{ match }* adalah objek yang diberikan oleh react-router-dom yang mengandung informasi tentang rute saat ini, termasuk parameter *URL*.

1. *Hook & State*

  const classes = useStyles()

  const [values, setValues] = useState({

    user: {following:[], followers:[]},

    redirectToSignin: false,

    following: false

  })

  const [posts, setPosts] = useState([])

  const jwt = auth.isAuthenticated()

* *useStyles()* adalah hook untuk mengakses kelas CSS.
* *useState* adalah hook untuk menyimpan dan mengelola state lokal di dalam komponen.
* *values* menyimpan informasi tentang pengguna, status *redirect*, dan apakah pengguna saat ini mengikuti profil ini.
* *posts* menyimpan daftar postingan pengguna.
* *jwt* menyimpan token autentikasi dari pengguna yang login.

1. *useEffect*

  useEffect(() => {

    const abortController = new AbortController()

    const signal = abortController.signal

    read({

      userId: match.params.userId

    }, {t: jwt.token}, signal).then((data) => {

      if (data && data.error) {

        setValues({...values, redirectToSignin: true})

      } else {

        let following = checkFollow(data)

        setValues({...values, user: data, following: following})

        loadPosts(data.\_id)

      }

    })

    return function cleanup(){

      abortController.abort()

    }

  }, [match.params.userId])

* *useEffect* adalah *hook* yang menjalankan kode setelah komponen dirender.
* *abortController* digunakan untuk membatalkan permintaan *fetch* jika komponen dilepas sebelum permintaan selesai.
* *read* adalah fungsi untuk mengambil data pengguna berdasarkan *userId* dari *URL*.
* Jika ada kesalahan, pengguna akan diarahkan ke halaman sign-in.
* Jika berhasil, data pengguna disimpan ke *values* dan fungsi *loadPosts* dipanggil untuk mengambil postingan pengguna.

1. *checkFollow*

  const checkFollow = (user) => {

*const checkFollow = (user) => {* mendefinisikan fungsi checkFollow yang menerima satu parameter user.

    const match = user.followers.some((follower)=> {

      return follower.\_id == jwt.user.\_id

    })

* *user.followers* adalah *array* yang berisi daftar pengikut (*followers*) dari pengguna.
* *some* adalah metode array yang menguji apakah setidaknya satu elemen dalam array memenuhi kriteria tertentu.
* *(follower) => { ... }* adalah fungsi *callback* yang akan dijalankan untuk setiap elemen dalam array followers. *follower.\_id* adalah *ID* dari pengikut.
* *jwt.user.\_id* adalah ID dari pengguna yang sedang login (diasumsikan bahwa informasi ini tersedia melalui objek jwt). Fungsi *callback return follower.\_id == jwt.user.\_id* adalah untuk mengembalikan true jika ID pengikut sama dengan ID pengguna yang sedang *login*, dan false jika tidak.

    return match

  }

*match* akan bernilai *true* jika setidaknya satu pengikut memiliki *ID* yang sama dengan *ID* pengguna yang sedang *login* (berarti pengguna tersebut mengikuti pengguna yang sedang *login*). Jika tidak ada pengikut yang memiliki *ID* yang sama, match akan bernilai *false*.

1. *clickFollowButton*

  const clickFollowButton = (callApi) => {

Deklarasi fungsi bernama *clickFollowButton* yang menerima satu parameter, yaitu *callApi*. *callApi* adalah fungsi yang akan dipanggil untuk menjalankan *API* terkait *follow/unfollow*.

    callApi({

      userId: jwt.user.\_id

    }, {

      t: jwt.token

    }, values.user.\_id).then((data) => {

      if (data.error) {

        setValues({...values, error: data.error})

      } else {

        setValues({...values, user: data, following: !values.following})

      }

    })

  }

*callApi* dipanggil dengan tiga argumen:

* Argumen pertama adalah objek yang berisi *userId*, diambil dari *jwt.user.\_id*.
* Argumen kedua adalah objek dengan token otentikasi *(t: jwt.token).*
* Argumen ketiga adalah *values.user.\_id*, yang merupakan *ID* pengguna yang akan *difollow* atau di-*unfollow*.

Setelah *callApi* dijalankan, fungsi *then* menangani respons dari *API*:

* Jika *data.error* ada, itu berarti ada kesalahan. Fungsi akan memperbarui state values dengan error yang diterima dari respons API.
* Jika tidak ada kesalahan, fungsi akan memperbarui *state* *values* dengan data pengguna terbaru (*user: data*) dan mengubah status following ke nilai kebalikannya *(!values.following).*

1. *loadPosts*

  const loadPosts = (user) => {

Deklarasi fungsi bernama *loadPosts*. Fungsi ini menerima satu parameter user, yang mewakili *ID* pengguna yang postingannya ingin kita ambil.

    listByUser({

      userId: user

    }, {

      t: jwt.token

    }).then((data) => {

      if (data.error) {

        console.log(data.error)

      } else {

        setPosts(data)

      }

    })

  }

* *listByUser* adalah fungsi yang di panggil untuk mendapatkan daftar postingan dari pengguna tertentu.
* Parameter pertama *{ userId: user }* adalah objek yang berisi *userId*, yang diisi dengan *ID* pengguna yang kita terima sebagai parameter *user*.
* Parameter kedua *{ t: jwt.token }* adalah objek lain yang berisi token otentikasi *(jwt.token),* digunakan untuk mengotorisasi permintaan ke *server*.
* *.then* digunakan untuk menangani hasil yang dikembalikan oleh *listByUser*. Setelah data diterima (berhasil atau tidak), fungsi yang didefinisikan dalam .then akan dieksekusi.
* *data* adalah *parameter* untuk fungsi ini, yang berisi hasil dari permintaan *listByUser*.
* Jika tidak ada kesalahan (*data.error* bernilai salah), dianggap bahwa data yang kita terima adalah daftar postingan.
* *setPosts(data)* adalah fungsi yang digunakan untuk memperbarui daftar postingan dengan data yang diterima.

1. *removePost*

  const removePost = (post) => {

Deklarasi fungsi bernama *removePost*. Fungsi ini menerima satu parameter post, yang mewakili postingan yang ingin kita hapus.

    const updatedPosts = posts

*updatedPosts* adalah variabel yang menyalin (referensi) daftar postingan saat ini yang disimpan dalam variabel *posts*. Penting untuk diingat bahwa ini tidak membuat salinan baru dari daftar, tetapi hanya membuat referensi ke daftar yang sama.

    const index = updatedPosts.indexOf(post)

*index* adalah variabel yang menyimpan posisi (indeks) dari postingan yang ingin dihapus dalam *updatedPosts*. *indexOf(post)* mencari posisi post dalam daftar *updatedPosts*.

    updatedPosts.splice(index, 1)

    setPosts(updatedPosts)

  }

*splice(index, 1)* adalah metode yang menghapus elemen dari daftar *updatedPosts* pada posisi *index*. 1 menunjukkan bahwa hanya ingin menghapus satu elemen dari posisi tersebut.

1. *photoURL*

    const photoUrl = values.user.\_id

              ? `/api/users/photo/${values.user.\_id}?${new Date().getTime()}`

              : '/api/users/defaultphoto'

*photoUrl* adalah *URL* untuk foto profil pengguna. Jika pengguna memiliki *\_id (values.user.\_id), URL* foto profil akan berisi *ID* pengguna dan timestamp untuk memastikan gambar terbaru diambil. Jika tidak, *URL* foto default *(/api/users/defaultphoto)* digunakan.

    if (values.redirectToSignin) {

      return <Redirect to='/signin'/>

    }

Jika *values.redirectToSignin* bernilai benar, komponen akan mengarahkan pengguna ke halaman masuk *(/signin).*

    return (

      <Paper className={classes.root} elevation={4}>

*Paper* adalah komponen dari *Material-UI* yang memberikan tampilan seperti kertas dengan bayangan *(elevation 4).*

        <Typography variant="h6" className={classes.title}>

          Profile

        </Typography>

*Typography* digunakan untuk menampilkan teks dengan gaya tertentu, dalam hal ini untuk judul "*Profile*".

        <List dense>

          <ListItem>

            <ListItemAvatar>

              <Avatar src={photoUrl} className={classes.bigAvatar}/>

            </ListItemAvatar>

            <ListItemText primary={values.user.name} secondary={values.user.email}/> {

             auth.isAuthenticated().user && auth.isAuthenticated().user.\_id == values.user.\_id

             ? (<ListItemSecondaryAction>

                  <Link to={"/user/edit/" + values.user.\_id}>

                    <IconButton aria-label="Edit" color="primary">

                      <Edit/>

                    </IconButton>

                  </Link>

                  <DeleteUser userId={values.user.\_id}/>

                </ListItemSecondaryAction>)

            : (<FollowProfileButton following={values.following} onButtonClick={clickFollowButton}/>)

            }

          </ListItem>

          <Divider/>

*<List>* digunakan untuk menampilkan daftar item secara vertikal. *<ListItem>* berisi detail pengguna, yaitu:

* *Avatar* menampilkan foto pengguna dari *photoUrl*.
* *ListItemText* menampilkan nama *(values.user.name)* dan email *(values.user.email)* pengguna.
* Jika pengguna yang sedang masuk *(auth.isAuthenticated().user.\_id)* adalah pengguna yang ditampilkan *(values.user.\_id),* aksi untuk mengedit dan menghapus profil muncul.
* Jika tidak, tombol untuk mengikuti/mengikuti ulang (*FollowProfileButton*) ditampilkan.

*<Divider>* adalah garis pemisah yang digunakan untuk memisahkan item dalam daftar.

          <ListItem>

            <ListItemText primary={values.user.about} secondary={"Joined: " + (

              new Date(values.user.created)).toDateString()}/>

          </ListItem>

        </List>

*<ListItemText>* menampilkan informasi "*About*" dari pengguna (*values.user.about*) dan tanggal bergabung (*values.user.created*).

        <ProfileTabs user={values.user} posts={posts} removePostUpdate={removePost}/>

      </Paper>

    )

}

*<ProfileTabs>* adalah komponen yang mungkin menampilkan tab untuk postingan dan aktivitas lain dari pengguna. *removePostUpdate={removePost}* memberikan fungsi *removePost* sebagai prop untuk memperbarui postingan ketika sebuah postingan dihapus. *</Paper>* menutup elemen Paper dan mengembalikan elemen *JSX* yang telah dibuat.

* My Profile Tabs

Source code untuk tampilan profile tabs (Posts, Following, dan Followers):

export default function ProfileTabs ( props ){

  const [tab, setTab] = useState(0)

  const handleTabChange = (event, value) => {

    setTab(value)

  }

    return (

    <div>

        <AppBar position="static" color="default">

          <Tabs

            value={tab}

            onChange={handleTabChange}

            indicatorColor="primary"

            textColor="primary"

            variant="fullWidth"

          >

            <Tab label="Posts" />

            <Tab label="Following" />

            <Tab label="Followers" />

          </Tabs>

        </AppBar>

       {tab === 0 && <TabContainer><PostList removeUpdate={props.removePostUpdate} posts={props.posts}/></TabContainer>}

       {tab === 1 && <TabContainer><FollowGrid people={props.user.following}/></TabContainer>}

       {tab === 2 && <TabContainer><FollowGrid people={props.user.followers}/></TabContainer>}

    </div>)

}

1. *ProfileTabs function*

export default function ProfileTabs ( props ){

  const [tab, setTab] = useState(0)

* *ProfileTabs* adalah komponen fungsional yang diekspor secara default.
* *useState(0)* digunakan untuk mendefinisikan *state* lokal tab yang diinisialisasi dengan nilai 0. *tab* akan melacak tab mana yang aktif (0 untuk *Posts*, 1 untuk *Following*, 2 untuk *Followers*).
* *setTab* adalah fungsi untuk memperbarui state tab.

1. *handleTabChange*

  const handleTabChange = (event, value) => {

    setTab(value)

  }

*handleTabChange* adalah fungsi yang dipanggil ketika pengguna mengklik salah satu *tab*. Fungsi ini menerima dua parameter: *event* (acara klik) dan *value* (nilai tab yang dipilih). *setTab(value)* memperbarui state tab dengan nilai tab yang dipilih.

1. *AppBar*

    return (

    <div>

        <AppBar position="static" color="default">

Komponen ini mengembalikan elemen *JSX* yang terdiri dari elemen div dan *AppBar*. *AppBar* adalah komponen dari *Material-UI* yang digunakan sebagai pembungkus untuk *tab*, dengan posisi statis dan warna default.

1. *Tabs*

          <Tabs

            value={tab}

            onChange={handleTabChange}

            indicatorColor="primary"

            textColor="primary"

            variant="fullWidth"

          >

            <Tab label="Posts" />

            <Tab label="Following" />

            <Tab label="Followers" />

          </Tabs>

*Tabs* adalah komponen dari *Material-UI* yang mengelola beberapa tab.

* *value={tab}* menentukan tab mana yang aktif berdasarkan state tab.
* *onChange={handleTabChange}* menentukan fungsi yang akan dipanggil ketika *tab* diubah.
* *indicatorColor, textColor*, dan *variant* digunakan untuk mengatur gaya *tab*.
* *Tab* adalah komponen untuk setiap tab dengan label yang sesuai: "*Posts*", "*Following*", dan "*Followers*".

        </AppBar>

       {tab === 0 && <TabContainer><PostList removeUpdate={props.removePostUpdate} posts={props.posts}/></TabContainer>}

       {tab === 1 && <TabContainer><FollowGrid people={props.user.following}/></TabContainer>}

       {tab === 2 && <TabContainer><FollowGrid people={props.user.followers}/></TabContainer>}

    </div>)

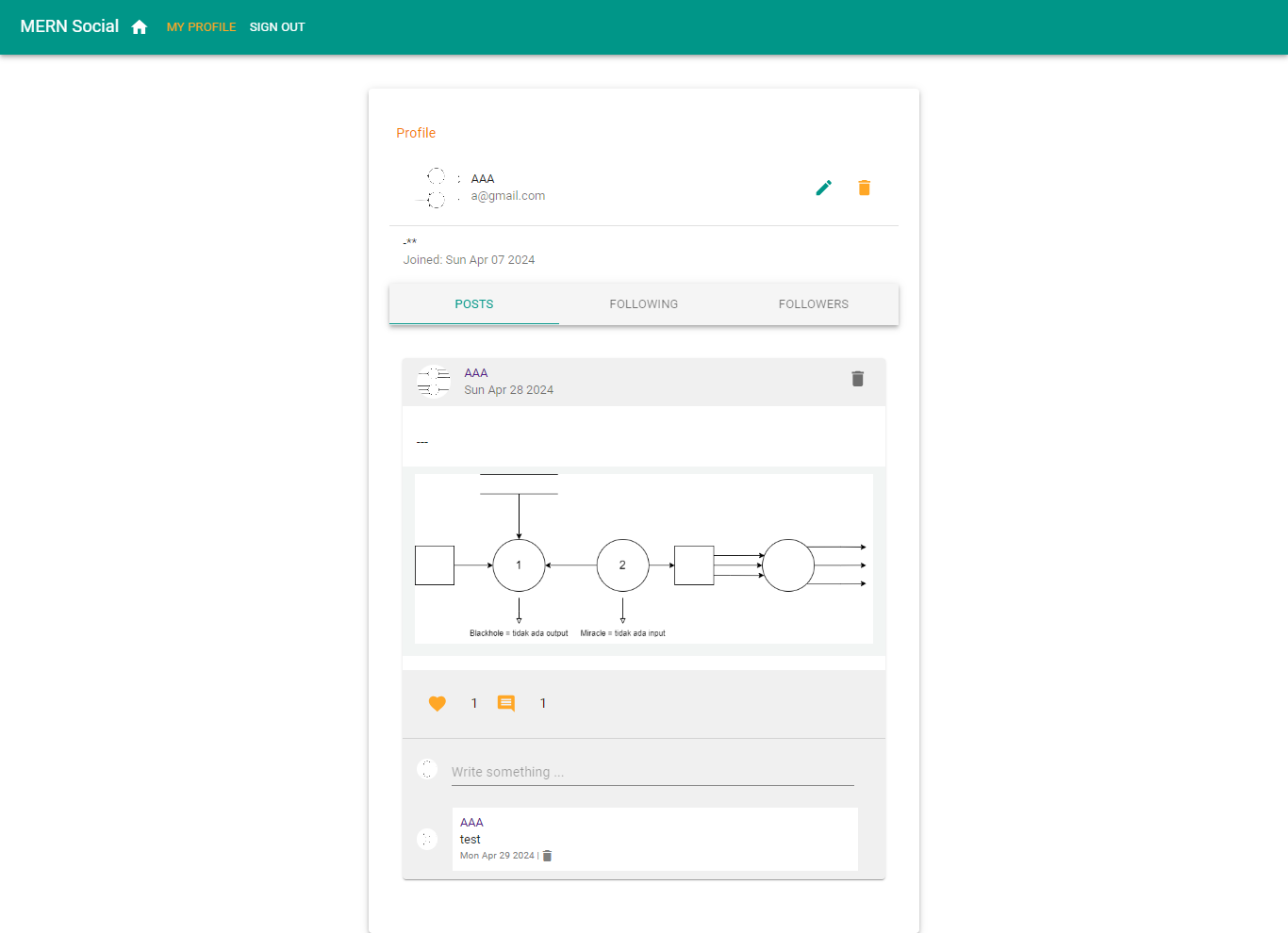
}

Setelah *AppBar*, ada kondisi untuk menampilkan konten berdasarkan tab yang aktif.

* Jika tab bernilai 0, *PostList* ditampilkan dalam *TabContainer* dengan *posts* dan *removeUpdate* dari *props*.
* Jika tab bernilai 1, *FollowGrid* untuk orang yang diikuti (*props.user.following*) ditampilkan dalam *TabContainer*.
* Jika tab bernilai 2, *FollowGrid* untuk pengikut (*props.user.followers*) ditampilkan dalam *TabContainer*.

1. Tab Post

Load the Posts



Source code untuk menampilkan semua post yang pengguna buat

const loadPosts = (user) => {

    listByUser(

      {

        userId: user,

      },

      {

        t: jwt.token,

      }

    ).then((data) => {

      if (data.error) {

        console.log(data.error);

      } else {

        setPosts(data);

      }

    });

  };

1. *loadPosts*

const loadPosts = (user) => {

Mendefinisikan fungsi *loadPosts* yang menerima satu parameter user. Parameter ini adalah *ID* pengguna yang postingannya ingin diambil.

    listByUser(

      {

        userId: user,

      },

      {

        t: jwt.token,

      }

*listByUser* adalah fungsi yang mengirim permintaan ke API untuk mendapatkan daftar postingan.

* Parameter pertama *{ userId: user }* adalah objek yang berisi ID pengguna (*userId*) yang diambil dari parameter user.
* Parameter kedua *{ t: jwt.token }* adalah objek yang berisi token otentikasi *(t)* yang digunakan untuk mengakses *API*. Token ini disimpan dalam *jwt.token*.

    ).then((data) => {

*.then* adalah metode yang digunakan untuk menangani hasil dari permintaan *listByUser*. Setelah *data* diterima, fungsi yang didefinisikan dalam *.then* akan dieksekusi. Parameter *data* adalah hasil dari permintaan *API*.

      if (data.error) {

        console.log(data.error);

      } else {

        setPosts(data);

      }

    });

  };

Mengecek apakah objek data memiliki properti *error*.

* Jika ada kesalahan (*data.error* bernilai benar), maka pesan kesalahan dicetak ke konsol menggunakan *console.log(data.error).*
* Jika tidak ada kesalahan (*data.error* bernilai salah), maka data yang diterima dianggap sebagai daftar postingan yang *valid* dan disimpan dalam *state* dengan memanggil *setPosts(data).*
* Trashcan icon (Remove a post from a list)

Source code untuk menghapus post dari *list*-nya

const removePost = (post) => {

    const updatedPosts = posts;

    const index = updatedPosts.indexOf(post);

    updatedPosts.splice(index, 1);

    setPosts(updatedPosts);

  };

<ProfileTabs

        user={values.user}

        posts={posts}

        removePostUpdate={removePost}

      />

1. *removePost*

const removePost = (post) => {

Mendefinisikan fungsi removePost yang menerima satu parameter post. Parameter ini adalah postingan yang ingin dihapus dari daftar.

    const updatedPosts = posts;

*updatedPosts* adalah variabel yang menyalin referensi dari daftar postingan yang ada (*posts*). Perlu diperhatikan bahwa ini bukan membuat salinan baru, melainkan hanya referensi ke daftar yang sama.

    const index = updatedPosts.indexOf(post);

*index* adalah variabel yang menyimpan posisi (indeks) dari postingan yang ingin dihapus dalam *updatedPosts*. *indexOf(post)* mencari posisi *post* dalam daftar *updatedPosts*.

    updatedPosts.splice(index, 1);

*splice(index, 1)* adalah metode yang menghapus satu elemen dari daftar *updatedPosts* pada posisi *index*.

    setPosts(updatedPosts);

  };

*setPosts(updatedPosts)* adalah fungsi yang digunakan untuk memperbarui state daftar postingan dengan *updatedPosts* yang telah dimodifikasi.

<ProfileTabs

        user={values.user}

        posts={posts}

        removePostUpdate={removePost}

      />

*ProfileTabs* adalah komponen yang menerima beberapa props: user, posts, dan removePostUpdate.

*removePostUpdate={removePost}* melewatkan fungsi *removePost* ke *ProfileTabs* melalui *prop* *removePostUpdate*.

1. FOLLOWING Tab

Source code yang menunjukkan semua akun yang pengguna telah follow:

export default function FollowGrid(props) {

  const classes = useStyles();

  return (

    <div className={classes.root}>

      <GridList cellHeight={160} className={classes.gridList} cols={4}>

        {props.people.map((person, i) => {

          return (

            <GridListTile style={{ height: 120 }} key={i}>

              <Link to={"/user/" + person.\_id}>

                <Avatar

                  src={"/api/users/photo/" + person.\_id}

                  className={classes.bigAvatar}

                />

                <Typography className={classes.tileText}>

                  {person.name}

                </Typography>

              </Link>

            </GridListTile>

          );

        })}

      </GridList>

    </div>

  );

}

FollowGrid.propTypes = {

  people: PropTypes.array.isRequired,

};

1. *FollowGrid*

export default function FollowGrid(props) {

  const classes = useStyles();

*FollowGrid* adalah komponen fungsional yang diekspor secara *default*. *useStyles* adalah *hook* yang digunakan untuk mengaplikasikan gaya *(CSS-in-JS)* menggunakan *Material-UI*.

  return (

    <div className={classes.root}>

Elemen *div* dengan kelas *root* yang diambil dari *useStyles*. Elemen *div* ini adalah elemen pembungkus utama untuk grid.

      <GridList cellHeight={160} className={classes.gridList} cols={4}>

*GridList* adalah komponen dari *Material-UI* yang digunakan untuk membuat grid.

* *cellHeight={160}* menentukan tinggi setiap sel dalam *grid*.
* *className={classes.gridList}* mengaplikasikan gaya *gridList* yang diambil dari *useStyles*.
* *cols={4}* menentukan jumlah kolom dalam *grid*.

        {props.people.map((person, i) => {

          return (

            <GridListTile style={{ height: 120 }} key={i}>

              <Link to={"/user/" + person.\_id}>

                <Avatar

                  src={"/api/users/photo/" + person.\_id}

                  className={classes.bigAvatar}

                />

                <Typography className={classes.tileText}>

                  {person.name}

                </Typography>

              </Link>

            </GridListTile>

          );

        })}

      </GridList>

    </div>

  );

}

* *props.people.map((person, i) => { ... })* memetakan setiap elemen dalam *array* people ke elemen *GridListTile*.
* *GridListTile* adalah komponen yang digunakan untuk setiap *item* dalam *grid*.
* *style={{ height: 120 }}* menetapkan tinggi setiap *tile* menjadi 120.
* *key={i}* memberikan kunci unik untuk setiap *tile* berdasarkan indeks.
* *Link* dari *react-router-dom* digunakan untuk membuat tautan ke halaman profil pengguna dengan *person.\_id.*
* *Avatar* menampilkan foto profil pengguna dengan *src={"/api/users/photo/" + person.\_id}* dan gaya *bigAvatar*.
* *Typography* menampilkan nama pengguna *(person.name)* dengan gaya *tileText*.

FollowGrid.propTypes = {

  people: PropTypes.array.isRequired,

};

*propTypes* digunakan untuk mendefinisikan tipe *prop* yang diterima oleh *FollowGrid*. *people* harus berupa *array* dan bersifat *wajib (isRequired).*

1. FOLLOWERS Tab

Source code yang menampilkan semua followers pengguna:

{tab === 2 && (

        <TabContainer>

          <FollowGrid people={props.user.followers} />

        </TabContainer>

      )}

1. *tab*

{tab === 2 && (

*tab === 2* menampilkan daftar pengikut menggunakan *FollowGrid*.

        <TabContainer>

          <FollowGrid people={props.user.followers} />

        </TabContainer>

      )}

*FollowGrid* menerima *people* sebagai *prop*, yang merupakan *array* dari objek pengguna.

* Menggunakan *GridList* dan *GridListTile* dari *Material-UI* untuk membuat grid pengguna.
* Setiap *GridListTile* berisi avatar dan nama pengguna, serta tautan ke halaman profil pengguna tersebut.
* *propTypes* memastikan bahwa *people* adalah *array* dan bersifat wajib (*isRequired*).

Source code untuk button *Follow* atau *Unfollow*

           (<FollowProfileButton following={values.following} onButtonClick={clickFollowButton}/>)

            }

Bagian ini menunjukkan bahwa kita sedang merender (menampilkan) komponen *React* bernama *FollowProfileButton*. Komponen ini menerima dua *props* (*parameter*) yang dikirim dari komponen induk (*parent component*):

* *following*: Nilai dari *values.following* yang menunjukkan apakah pengguna saat ini sudah mengikuti pengguna lain atau belum.
* *onButtonClick*: Fungsi *clickFollowButton* yang akan dipanggil ketika tombol diklik.

Source code yang menjalankan *button*-nya

export default function FollowProfileButton (props) {

  const followClick = () => {

    props.onButtonClick(follow)

  }

  const unfollowClick = () => {

    props.onButtonClick(unfollow)

  }

    return (<div>

      { props.following

        ? (<Button variant="contained" color="secondary" onClick={unfollowClick}>Unfollow</Button>)

        : (<Button variant="contained" color="primary" onClick={followClick}>Follow</Button>)

      }

    </div>)

}

FollowProfileButton.propTypes = {

  following: PropTypes.bool.isRequired,

  onButtonClick: PropTypes.func.isRequired

}

1. *FollowProfileButton*

export default function FollowProfileButton (props) {

Mendefinisikan komponen *FollowProfileButton* dan mengekspornya sebagai *default*. Komponen ini menerima *props* sebagai parameter

  const followClick = () => {

    props.onButtonClick(follow)

  }

  const unfollowClick = () => {

    props.onButtonClick(unfollow)

  }

* *followClick*: Fungsi ini dipanggil saat tombol "*Follow*" diklik. Ini memanggil props.onButtonClick dengan argumen *follow*.
* *unfollowClick*: Fungsi ini dipanggil saat tombol "*Unfollow*" diklik. Ini memanggil *props.onButtonClick* dengan argumen *unfollow*.

    return (<div>

      { props.following

        ? (<Button variant="contained" color="secondary" onClick={unfollowClick}>Unfollow</Button>)

        : (<Button variant="contained" color="primary" onClick={followClick}>Follow</Button>)

      }

    </div>)

}

*props.following* digunakan untuk menentukan tombol mana yang akan ditampilkan.

* Jika *props.following* adalah *true*, tombol "*Unfollow*" ditampilkan.
* Jika *props.following* adalah *false*, tombol "*Follow*" ditampilkan.

FollowProfileButton.propTypes = {

  following: PropTypes.bool.isRequired,

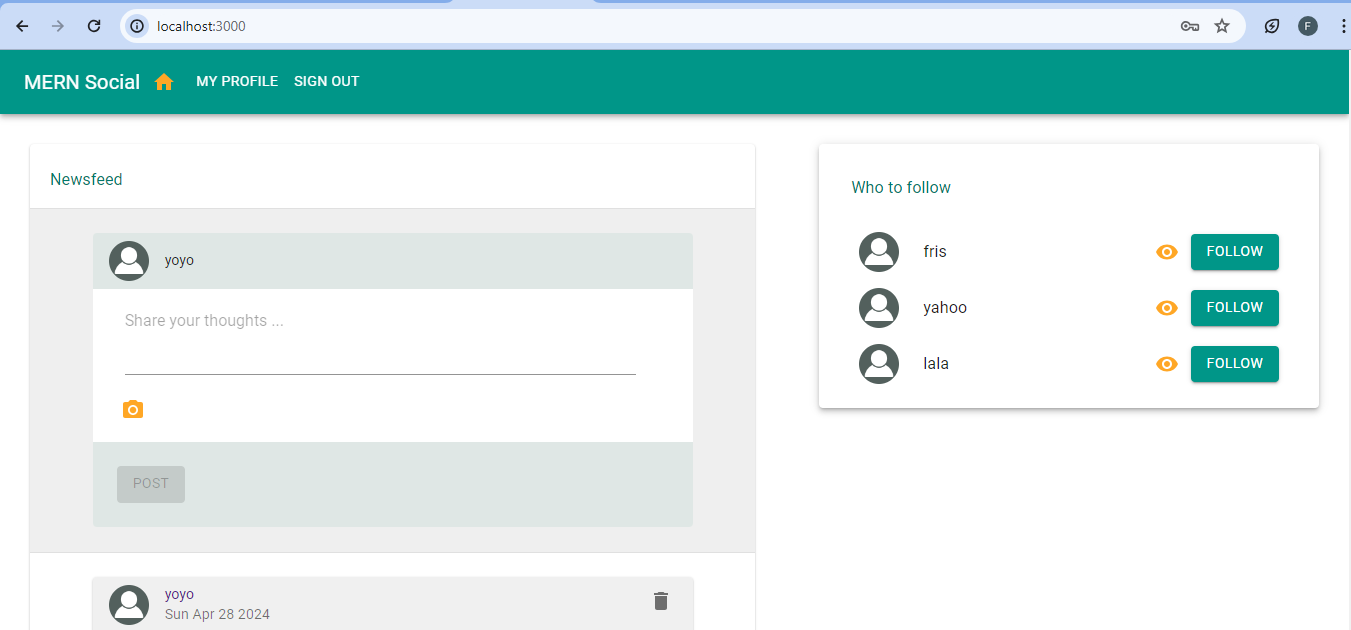
  onButtonClick: PropTypes.func.isRequired

}

Mendefinisikan tipe properti (*props*) yang diharapkan oleh komponen ini.

* *following*: Harus berupa nilai *boolean* (*true* atau *false*).
* *onButtonClick*: Harus berupa fungsi.

### F. Halaman Beranda



Pada halaman ini terdapat fitur “My Profile”, “Sign Out”, “Post”, “Follow”.

* **Post button**

Source code yang menjalankan *button* *POST:*

export default function NewPost (props){

  const classes = useStyles()

  const [values, setValues] = useState({

    text: '',

    photo: '',

    error: '',

    user: {}

  })

  const jwt = auth.isAuthenticated()

  useEffect(() => {

    setValues({...values, user: auth.isAuthenticated().user})

  }, [])

  const clickPost = () => {

    let postData = new FormData()

    postData.append('text', values.text)

    postData.append('photo', values.photo)

    create({

      userId: jwt.user.\_id

    }, {

      t: jwt.token

    }, postData).then((data) => {

      if (data.error) {

        setValues({...values, error: data.error})

      } else {

        setValues({...values, text:'', photo: ''})

        props.addUpdate(data)

      }

    })

  }

  const handleChange = name => event => {

    const value = name === 'photo'

      ? event.target.files[0]

      : event.target.value

    setValues({...values, [name]: value })

  }

  const photoURL = values.user.\_id ?'/api/users/photo/'+ values.user.\_id : '/api/users/defaultphoto'

    return (<div className={classes.root}>

      <Card className={classes.card}>

      <CardHeader

            avatar={

              <Avatar src={photoURL}/>

            }

            title={values.user.name}

            className={classes.cardHeader}

          />

      <CardContent className={classes.cardContent}>

        <TextField

            placeholder="Share your thoughts ..."

            multiline

            rows="3"

            value={values.text}

            onChange={handleChange('text')}

            className={classes.textField}

            margin="normal"

        />

        <input accept="image/\*" onChange={handleChange('photo')} className={classes.input} id="icon-button-file" type="file" />

        <label htmlFor="icon-button-file">

          <IconButton color="secondary" className={classes.photoButton} component="span">

            <PhotoCamera />

          </IconButton>

        </label> <span className={classes.filename}>{values.photo ? values.photo.name : ''}</span>

        { values.error && (<Typography component="p" color="error">

            <Icon color="error" className={classes.error}>error</Icon>

              {values.error}

            </Typography>)

        }

      </CardContent>

      <CardActions>

        <Button color="primary" variant="contained" disabled={values.text === ''} onClick={clickPost} className={classes.submit}>POST</Button>

      </CardActions>

    </Card>

  </div>)

}

NewPost.propTypes = {

  addUpdate: PropTypes.func.isRequired

}

1. NewPost function

export default function NewPost (props){

  const classes = useStyles()

  const [values, setValues] = useState({

    text: '',

    photo: '',

    error: '',

    user: {}

  })

  const jwt = auth.isAuthenticated()

* Mengimpor fungsi *NewPost* sebagai komponen *default*.
* *useStyles*: *Hook* untuk menggunakan kelas CSS.
* *useState*: *Hook* untuk mendefinisikan *state* lokal komponen.
* *auth.isAuthenticated():* Mendapatkan informasi autentikasi pengguna.

1. *useEffect*

  useEffect(() => {

    setValues({...values, user: auth.isAuthenticated().user})

  }, [])

* *useEffect*: Hook yang menjalankan kode saat komponen pertama kali dirender.
* *setValues* mengatur nilai *user* dalam *state* *values* dengan data pengguna yang sedang *login*.

1. *clickPost*

  const clickPost = () => {

    let postData = new FormData()

    postData.append('text', values.text)

    postData.append('photo', values.photo)

    create({

      userId: jwt.user.\_id

    }, {

      t: jwt.token

    }, postData).then((data) => {

      if (data.error) {

        setValues({...values, error: data.error})

      } else {

        setValues({...values, text:'', photo: ''})

        props.addUpdate(data)

      }

    })

  }

* *clickPost*: Fungsi yang dipanggil saat tombol *POST* diklik.
* Membuat objek *FormData* dan menambahkan teks dan foto dari state *values*.
* Memanggil fungsi *create* untuk mengirim *data* ke *server*.
* Jika ada kesalahan, *error* diatur dalam *state* *values*. Jika berhasil, mengosongkan teks dan foto dan memanggil fungsi *addUpdate* dari *props*.

1. *handleChange*

  const handleChange = name => event => {

    const value = name === 'photo'

      ? event.target.files[0]

      : event.target.value

    setValues({...values, [name]: value })

  }

* *handleChange*: Mengelola perubahan *input* pada teks dan foto.
* Jika *input* adalah foto, *file* diambil dari *event.target.files[0].* Jika input adalah *teks*, nilai diambil dari *event.target.value*.
* Mengatur nilai yang sesuai dalam *state values*.

1. *photoURL*

  const photoURL = values.user.\_id ?'/api/users/photo/'+ values.user.\_id : '/api/users/defaultphoto'

*const photoURL* menentukan URL foto pengguna. Jika pengguna memiliki ID, URL foto pengguna digunakan. Jika tidak, URL foto default digunakan.

1. Render komponen *posting*

    return (<div className={classes.root}>

      <Card className={classes.card}>

      <CardHeader

            avatar={

              <Avatar src={photoURL}/>

            }

            title={values.user.name}

            className={classes.cardHeader}

          />

      <CardContent className={classes.cardContent}>

        <TextField

            placeholder="Share your thoughts ..."

            multiline

            rows="3"

            value={values.text}

            onChange={handleChange('text')}

            className={classes.textField}

            margin="normal"

        />

        <input accept="image/\*" onChange={handleChange('photo')} className={classes.input} id="icon-button-file" type="file" />

        <label htmlFor="icon-button-file">

          <IconButton color="secondary" className={classes.photoButton} component="span">

            <PhotoCamera />

          </IconButton>

        </label>

<span className={classes.filename}>{values.photo ? values.photo.name : ''}</span>

Menggunakan komponen *Material-UI* untuk membuat elemen form yang telah di-*style* dengan elemen *Card*, *CardHeader*, *CardContent*, *TextField*, *IconButton*, dan *Button* untuk membentuk struktur form.

* *Avatar*: Menampilkan foto pengguna.
* *TextField*: Input teks untuk postingan.
* *input* dan *IconButton*: *Input* untuk mengunggah foto.

        { values.error && (<Typography component="p" color="error">

            <Icon color="error" className={classes.error}>error</Icon>

              {values.error}

            </Typography>)

        }

      </CardContent>

Menampilkan pesan kesalahan jika ada.

* *{values.error && (...)}* adalah cara untuk membuat elemen *JSX* secara kondisional. Jika *values.error* memiliki nilai (yaitu, ada kesalahan), maka elemen di dalam tanda kurung akan dirender.
* *<Typography component="p" color="error">:* Komponen *Typography* dari *Material-UI* yang menampilkan teks dengan warna kesalahan. *component="p"* berarti elemen ini akan dirender sebagai elemen paragraf *<p>.*
* *<Icon color="error" className={classes.error}>error</Icon>:* Menampilkan ikon kesalahan dari *Material-UI* dengan warna kesalahan. *className={classes.error}* memberikan gaya CSS khusus pada ikon ini.
* *{values.error}* menampilkan pesan kesalahan yang disimpan dalam *values.error*.

      <CardActions>

        <Button color="primary" variant="contained" disabled={values.text === ''} onClick={clickPost} className={classes.submit}>POST</Button>

      </CardActions>

    </Card>

  </div>)

}

*CardActions*: Komponen ini digunakan untuk mengatur elemen aksi, seperti tombol, di bagian bawah kartu.

* *Button*: Tombol yang memungkinkan pengguna untuk mengirim postingan baru.
* *color="primary":* Memberikan warna utama pada tombol.
* *variant="contained":* Membuat tombol tampak lebih tebal dan menonjol dengan latar belakang berwarna.
* *disabled={values.text === ''}:* Menonaktifkan tombol jika *values.text* kosong, sehingga pengguna tidak bisa mengirim postingan tanpa teks.
* *onClick={clickPost}:* Menentukan fungsi *clickPost* untuk dijalankan saat tombol diklik.
* *className={classes.submit}:* Memberikan kelas CSS khusus pada tombol untuk penataan.

NewPost.propTypes = {

  addUpdate: PropTypes.func.isRequired

}

*PropTypes* adalah library di *React* yang digunakan untuk memeriksa tipe data properti yang diteruskan ke komponen. Ini membantu dalam menangkap *bug* dengan memastikan bahwa komponen menerima data dengan tipe yang benar.

* *NewPost.propTypes* mendefinisikan tipe properti yang diharapkan oleh komponen *NewPost*.
* *addUpdate*: Nama properti yang diharapkan.
* *PropTypes.func*: Menunjukkan bahwa *addUpdate* harus berupa fungsi.
* *isRequired*: Menunjukkan bahwa *addUpdate* adalah properti wajib. Jika properti ini tidak disediakan, *React* akan mengeluarkan peringatan.
* Camera button

Source code yang menjalankan icon *Camera* untuk membuka *File Manager* pada tampilan pengguna:

        <input accept="image/\*" onChange={handleChange('photo')} className={classes.input} id="icon-button-file" type="file" />

<label htmlFor="icon-button-file">

          <IconButton color="secondary" className={classes.photoButton} component="span">

            <PhotoCamera />

          </IconButton>

        </label> <span className={classes.filename}>{values.photo ? values.photo.name : ''}</span>

        { values.error && (<Typography component="p" color="error">

            <Icon color="error" className={classes.error}>error</Icon>

              {values.error}

            </Typography>)

        }

      </CardContent>

1. Elemen *input*

        <input accept="image/\*" onChange={handleChange('photo')} className={classes.input} id="icon-button-file" type="file" />

* *<input>:* Elemen HTML yang digunakan untuk memilih file.
* *accept="image/\*":* Hanya menerima file gambar (seperti *.jpg, .png*).
* *onChange={handleChange('photo')}:* Fungsi yang akan dipanggil saat pengguna memilih *file*. Ini mengupdate *state* komponen dengan file yang dipilih.
* *className={classes.input}:* Menambahkan kelas CSS untuk styling.
* *id="icon-button-file":* Memberikan ID unik pada elemen ini.

<label htmlFor="icon-button-file">

          <IconButton color="secondary" className={classes.photoButton} component="span">

            <PhotoCamera />

          </IconButton>

</label>

* *<label>:* Label untuk *input* file. Klik label ini sama dengan klik input file karena mereka terhubung *oleh htmlFor="icon-button-file".*
* *<IconButton>:* Tombol dari *Material-UI* yang digunakan untuk styling dan ikon.
* *color="secondary":* Mengatur warna tombol menjadi warna sekunder.
* *className={classes.photoButton}:* Menambahkan kelas *CSS* untuk styling.
* *component="span":* Membuat tombol ini berperilaku seperti elemen *<span>.*
* *<PhotoCamera />:* Ikon kamera dari Material-UI.

        <span className={classes.filename}>{values.photo ? values.photo.name : ''}</span>

* *<span>:* Elemen HTML untuk menampilkan teks.
* *className={classes.filename}:* Menambahkan kelas CSS untuk styling.
* *{values.photo ? values.photo.name : ''}:* Jika ada file yang dipilih *(values.photo),* tampilkan nama file *(values.photo.name).* Jika tidak, tampilkan *string* kosong.

        { values.error && (<Typography component="p" color="error">

            <Icon color="error" className={classes.error}>error</Icon>

              {values.error}

            </Typography>)

        }

      </CardContent>

* *values.error && (...):* Jika ada error (*values.error*), tampilkan pesan *error*.
* *<Typography>:* Komponen dari *Material-UI* untuk teks yang diberi gaya.
* *component="p":* Membuatnya berperilaku seperti elemen *<p>.*
* *color="error":* Mengatur warna teks menjadi warna *error*.
* *<Icon>:* Komponen dari *Material-UI* untuk ikon.
* *className={classes.error}:* Menambahkan kelas *CSS* untuk *styling*.
* *{values.error}:* Menampilkan pesan *error*.
* </CardContent> penutup dari CardContent yang mengindikasikan akhir dari konten kartu.
* Delete post button

Source code yang menjalankan icon Delete comment:

const deletePost = () => {

    remove({

      postId: props.post.\_id

    }, {

      t: jwt.token

    }).then((data) => {

      if (data.error) {

        console.log(data.error)

      } else {

        props.onRemove(props.post)

      }

    })

  }

1. deletePost

const deletePost = () => {

*const deletePost = () => {}:* Mendefinisikan fungsi *deletePost* sebagai fungsi panah (*arrow function*) tanpa parameter.

    remove({

      postId: props.post.\_id

    }, {

      t: jwt.token

    }).then((data) => {

*remove*: sebuah fungsi yang dipanggil untuk menghapus postingan dari *server* atau *database*. *remove* menerima dua argumen:

* *{ postId: props.post.\_id }:* Objek yang berisi *postId*, yaitu *ID* dari postingan yang ingin dihapus. *props.post.\_id* adalah *ID* dari postingan yang diambil dari properti *props*.
* *{ t: jwt.token }:* Objek yang berisi token autentikasi *(jwt.token),* digunakan untuk memastikan bahwa pengguna yang menghapus postingan memiliki izin yang benar.

*.then((data) => { ... }):* Setelah fungsi *remove* selesai dan mengembalikan hasil, *.then* digunakan untuk menangani hasil tersebut. *data* adalah hasil dari operasi penghapusan.

      if (data.error) {

        console.log(data.error)

      } else {

        props.onRemove(props.post)

      }

    })

  }

* *if (data.error):* Mengecek apakah ada *error* dalam hasil (*data*).
* *console.log(data.error):* Jika ada error, mencetak pesan *error* ke konsol.
* *else*: Jika tidak ada error.
* *props.onRemove(props.post)*: Memanggil fungsi *onRemove* yang diterima dari properti *props*. Fungsi ini biasanya digunakan untuk menghapus postingan dari tampilan pengguna (*UI*).

Terdapat juga fungsi deletePost yang digunakan untuk menghapus postingan. Ini hanya dapat dilakukan oleh pengguna yang mempostingnya sendiri. Setelah postingan dihapus, properti onRemove yang diterima dari luar akan dipanggil untuk memberitahu komponen induk bahwa postingan telah dihapus.

* Like button

Source code yang memanggil *request PUT* agar suatu post dapat di-*like*, source code ini diletakkan pada *post.routes.js* :

router.route('/api/posts/like')

  .put(authCtrl.requireSignin, postCtrl.like)

1. *Request PUT Like*

router.route('/api/posts/like')

*router.route('/api/posts/like')* mendefinisikan rute untuk *endpoint /api/posts/like.* Ini berarti bahwa *URL* ini akan menangani permintaan yang berhubungan dengan "*like*" pada postingan.

  .put(authCtrl.requireSignin, postCtrl.like)

*.put(...)* mengindikasikan bahwa rute ini akan menangani permintaan *HTTP* *PUT*. Metode *PUT* biasanya digunakan untuk memperbarui *data* di *server*. Dalam kode ini, *PUT* digunakan untuk menambahkan "*like*" pada postingan.

*authCtrl.requireSignin* adalah middleware. *Middleware* adalah fungsi yang berjalan sebelum fungsi utama untuk menangani permintaan. *requireSignin* mungkin memeriksa apakah pengguna sudah masuk (autentikasi) sebelum mengizinkan mereka untuk memberikan "*like*" pada postingan.

*postCtrl.like* adalah fungsi kontroler yang akan dijalankan jika pengguna telah berhasil melewati *middleware* *requireSignin*. Fungsi ini akan menangani logika untuk menambahkan "*like*" pada postingan.

Agar program menemukan *post* yang di-*like* oleh pengguna, digunakan *$push, source code* ini ada di *post.controller.js* :

const like = async (req, res) => {

  try{

    let result = await Post.findByIdAndUpdate(req.body.postId, {$push: {likes: req.body.userId}}, {new: true})

    res.json(result)

  }catch(err){

      return res.status(400).json({

        error: errorHandler.getErrorMessage(err)

      })

  }

}

1. *$push Like*

const like = async (req, res) => {

* *const like = async (req, res) => {}:* Mendefinisikan fungsi like sebagai fungsi asinkron (dengan kata kunci *async*). Ini berarti fungsi ini bisa menggunakan await di dalamnya.
* *req*: Singkatan dari request, ini adalah objek yang berisi informasi tentang permintaan *HTTP* yang diterima.
* *res*: Singkatan dari *response*, ini adalah objek yang digunakan untuk mengirim respons kembali ke klien.

  try{

    let result = await Post.findByIdAndUpdate(req.body.postId, {$push: {likes: req.body.userId}}, {new: true})

    res.json(result)

*try { ... }:* adalah bagian yang mencoba menjalankan kode di dalamnya. Jika ada kesalahan (*error*) yang terjadi, kontrol akan berpindah ke blok *catch*.

* *let result = await ...: await* digunakan untuk menunggu operasi asinkron selesai sebelum melanjutkan eksekusi kode berikutnya. result akan menyimpan hasil dari operasi ini.
* *Post.findByIdAndUpdate(...)* adalah metode dari model *Post* untuk mencari postingan berdasarkan *ID* *(req.body.postId)* dan memperbaruinya.
* *req.body.postId: ID* dari postingan yang akan di-"*like*" diambil dari body permintaan.
* *{ $push: { likes: req.body.userId } }:* Operasi pembaruan menggunakan operator *MongoDB $push* untuk menambahkan *userId* ke array likes dalam postingan tersebut.
* *{ new: true }:* Opsi ini memastikan bahwa nilai yang dikembalikan adalah dokumen yang sudah diperbarui.
* *res.json(result):* Mengirim respons kembali ke klien dengan hasil pembaruan (postingan yang sudah di-"*like*").

  }catch(err){

      return res.status(400).json({

        error: errorHandler.getErrorMessage(err)

      })

  }

}

* *catch (err) { ... }:* Jika ada kesalahan yang terjadi dalam blok *try*, blok ini akan menangkap kesalahan tersebut.
* *return res.status(400).json({ ... }):* Mengirim respons dengan status *HTTP 400 (Bad Request)* dan sebuah objek *JSON* yang berisi pesan error.
* *errorHandler.getErrorMessage(err):* Fungsi *getErrorMessage* dari *errorHandler* digunakan untuk mendapatkan pesan *error* yang lebih jelas dari objek *error* *err*.

Source code untuk metode *fetch like*, terletak di *api-post.js*:

const like = async (params, credentials, postId) => {

  try {

    let response = await fetch("/api/posts/like/", {

      method: "PUT",

      headers: {

        Accept: "application/json",

        "Content-Type": "application/json",

        Authorization: "Bearer " + credentials.t,

      },

      body: JSON.stringify({ userId: params.userId, postId: postId }),

    });

    return await response.json();

  } catch (err) {

    console.log(err);

  }

};

1. *Fetch Like*

const like = async (params, credentials, postId) => {

* *const like = async (params, credentials, postId) => {}:* Mendefinisikan fungsi like sebagai fungsi asinkron (dengan kata kunci *async*). Ini berarti fungsi ini bisa menggunakan *await* di dalamnya.
* *params, credentials, postId*: Parameter yang diterima oleh fungsi:
* *params*: Berisi data terkait pengguna, misalnya *params.userId* adalah *ID* pengguna yang memberikan "*like*".
* *credentials*: Berisi informasi autentikasi, misalnya *credentials*.*t* adalah token autentikasi.
* *postId*: *ID* dari postingan yang akan di-"*like*".

  try {

    let response = await fetch("/api/posts/like/", {

      method: "PUT",

      headers: {

        Accept: "application/json",

        "Content-Type": "application/json",

        Authorization: "Bearer " + credentials.t,

      },

      body: JSON.stringify({ userId: params.userId, postId: postId }),

    });

* *try { ... }:* Bagian ini mencoba menjalankan kode di dalamnya. Jika ada kesalahan (*error*) yang terjadi, kontrol akan berpindah ke blok catch.
* *let response = await fetch(...):* *fetch* digunakan untuk mengirim permintaan *HTTP*. *await* digunakan untuk menunggu respons dari *server* sebelum melanjutkan eksekusi kode berikutnya.
* *"/api/posts/like/":* *URL endpoint* di *server* yang akan menangani permintaan untuk menambahkan "*like*" pada postingan.
* *method: "PUT":* Menggunakan metode *HTTP* *PUT* untuk memperbarui data di *server*.
* *headers*: { ... }: Menyertakan beberapa *header* dalam permintaan:
* *Accept: "application/json":* Memberitahu *server* bahwa klien mengharapkan respons dalam format *JSON*.
* *"Content-Type": "application/json":* Memberitahu *server* bahwa *data* yang dikirim dalam permintaan adalah *JSON*.
* *Authorization: "Bearer " + credentials.t:* Menyertakan *token* autentikasi dalam *header* *Authorization*.
* *body: JSON.stringify({ userId: params.userId, postId: postId }):* Menyertakan data dalam body permintaan sebagai *JSON*. Data ini berisi *userId* (*ID* pengguna yang memberikan "*like*") dan *postId* (*ID* dari postingan yang akan di-"*like*").

    return await response.json();

  } catch (err) {

    console.log(err);

  }

};

* *return await response.json():* Menunggu respons dari *server* dalam format *JSON* dan mengembalikannya dari fungsi *like*.
* *catch (err) { ... }:* Jika ada kesalahan yang terjadi dalam blok *try*, blok ini akan menangkap kesalahan tersebut.
* *console.log(err):* Mencetak kesalahan ke konsol untuk *debugging*.

Source code untuk menangani klik pada icon *like,* yang akan memanggil metode fetch yang sesuai untuk meng-update *like* dan hitungan *like* :

const clickLike = () => {

    let callApi = values.like ? unlike : like;

    callApi(

      {

        userId: jwt.user.\_id,

      },

      {

        t: jwt.token,

      },

      props.post.\_id

    ).then((data) => {

      if (data.error) {

        console.log(data.error);

      } else {

        setValues({ ...values, like: !values.like, likes: data.likes.length });

      }

    });

  };

1. clickLike

const clickLike = () => {

*const clickLike = () => {}:* Mendefinisikan fungsi *clickLike* sebagai fungsi panah (*arrow function*) tanpa parameter.

    let callApi = values.like ? unlike : like;

*let callApi = values.like ? unlike : like;:* Menentukan apakah akan memanggil fungsi unlike atau like berdasarkan status "*like*" saat ini.

* Jika *values.like* adalah *true* (postingan saat ini sudah di-*like*), maka *callApi* akan diatur ke fungsi *unlike*.
* Jika *values.like* adalah *false* (postingan saat ini belum di-*like*), maka *callApi* akan diatur ke fungsi *like*.

    callApi(

      {

        userId: jwt.user.\_id,

      },

      {

        t: jwt.token,

      },

      props.post.\_id

)

*callApi({ userId: jwt.user.\_id }, { t: jwt.token }, props.post.*\_id): Memanggil fungsi *callApi* (yang bisa jadi *unlike* atau *like*) dengan tiga argumen:

* *{ userId: jwt.user.\_id }:* Objek yang berisi *ID* pengguna yang sedang *login*.
* *{ t: jwt.token }:* Objek yang berisi token autentikasi.
* *props.post.\_id*: *ID* dari postingan yang sedang di-*like* atau di-*unlike*.

.then((data) => {

      if (data.error) {

        console.log(data.error);

      } else {

        setValues({ ...values, like: !values.like, likes: data.likes.length });

      }

    });

  };

*.then((data) => { ... }):* Setelah *API* dipanggil dan respons diterima, fungsi ini menangani hasilnya:

* *if (data.error) { console.log(data.error); }:* Jika ada error dalam respons, cetak *error* ke konsol.
* *else { setValues({ ...values, like: !values.like, likes: data.likes.length }); }:* Jika tidak ada *error*, perbarui nilai *values*:
  + *like: !values.like:* Membalik nilai *like*. Jika sebelumnya *true*, menjadi *false*, dan sebaliknya.
  + *likes: data.likes.length:* Mengatur jumlah "*like*" berdasarkan jumlah "*like*" yang baru dari respons.

Fungsi *clickLike* digunakan untuk menangani klik tombol suka. Jika pengguna sudah menyukai postingan, maka fungsi unlike akan dipanggil; jika belum, maka fungsi like akan dipanggil. Setelah itu, jumlah suka akan diperbarui berdasarkan respons dari server.

* Unlike button

Source code yang memanggil *request* *PUT* untuk aksi unlike:

router.route('/api/posts/unlike')

  .put(authCtrl.requireSignin, postCtrl.unlike)

1. Request *PUT Unlike*

router.route('/api/posts/unlike')

*router.route('/api/posts/unlike')* mendefinisikan rute untuk *endpoint* */api/posts/unlike*. Ini berarti bahwa *URL* ini akan menangani permintaan yang berhubungan dengan "*unlike*" pada postingan.

  .put(authCtrl.requireSignin, postCtrl.unlike)

* *.put(...)* mengindikasikan bahwa rute ini akan menangani permintaan *HTTP PUT*. Metode *PUT* biasanya digunakan untuk memperbarui data di server. Dalam konteks ini, kita menggunakannya untuk menghapus "*like*" dari postingan.
* *authCtrl.requireSignin* adalah middleware. *Middleware* adalah fungsi yang berjalan sebelum fungsi utama untuk menangani permintaan. requireSignin mungkin memeriksa apakah pengguna sudah masuk (autentikasi) sebelum mengizinkan mereka untuk menghapus "*like*" dari postingan.
* *postCtrl.unlike* adalah fungsi kontroler yang akan dijalankan jika pengguna telah berhasil melewati *middleware* *requireSignin*. Fungsi ini akan menangani logika untuk menghapus "*like*" dari postingan.

Menggunakan metode *controller* untuk mencari post dan meng-update array like dengan menghapus *ID* *user*-nya menggunakan *$pull* :

const unlike = async (req, res) => {

  try{

    let result = await Post.findByIdAndUpdate(req.body.postId, {$pull: {likes: req.body.userId}}, {new: true})

    res.json(result)

  }catch(err){

    return res.status(400).json({

      error: errorHandler.getErrorMessage(err)

    })

  }

}

1. *$pull Unlike*

const unlike = async (req, res) => {

*const unlike = async (req, res) => {}:* Mendefinisikan fungsi *unlike* sebagai fungsi asinkron (dengan kata kunci *async*). Ini berarti fungsi ini bisa menggunakan await di dalamnya.

* *req*: Singkatan dari *request*, ini adalah objek yang berisi informasi tentang permintaan *HTTP* yang diterima.
* *res*: Singkatan dari *response*, ini adalah objek yang digunakan untuk mengirim respons kembali ke klien.

  try{

    let result = await Post.findByIdAndUpdate(req.body.postId, {$pull: {likes: req.body.userId}}, {new: true})

    res.json(result)

* *try { ... }:* Bagian ini mencoba menjalankan kode di dalamnya. Jika ada kesalahan (error) yang terjadi, kontrol akan berpindah ke blok *catch*.
* *let result = await ...:* await digunakan untuk menunggu operasi asinkron selesai sebelum melanjutkan eksekusi kode berikutnya. result akan menyimpan hasil dari operasi ini.
* *Post.findByIdAndUpdate(...):* Ini adalah metode dari *model* *Post* untuk mencari postingan berdasarkan ID (*req.body.postId*) dan memperbaruinya.
* *req.body.postId: ID* dari postingan yang akan di-"*unlike*" diambil dari body permintaan.
* *{ $pull: { likes: req.body.userId } }:* Operasi pembaruan menggunakan operator *MongoDB $pull* untuk menghapus *userId* dari array *likes* dalam postingan tersebut.
* *{ new: true }:* Opsi ini memastikan bahwa nilai yang dikembalikan adalah dokumen yang sudah diperbarui.
* *res.json(result):* Mengirim respons kembali ke klien dengan hasil pembaruan (postingan yang sudah di-unlike).

  }catch(err){

    return res.status(400).json({

      error: errorHandler.getErrorMessage(err)

    })

  }

}

* *catch (err) { ... }:* Jika ada kesalahan yang terjadi dalam blok *try*, blok ini akan menangkap kesalahan tersebut.
* *return res.status(400).json({ ... }):* Mengirim respons dengan status *HTTP 400 (Bad Request)* dan sebuah objek *JSON* yang berisi pesan error.
* *errorHandler.getErrorMessage(err):* Fungsi *getErrorMessage* dari *errorHandler* digunakan untuk mendapatkan pesan error yang lebih jelas dari objek error *err*.

Metode fetch unlike yang terletak di file *api-post.js* :

const unlike = async (params, credentials, postId) => {

  try {

    let response = await fetch("/api/posts/unlike/", {

      method: "PUT",

      headers: {

        Accept: "application/json",

        "Content-Type": "application/json",

        Authorization: "Bearer " + credentials.t,

      },

      body: JSON.stringify({ userId: params.userId, postId: postId }),

    });

    return await response.json();

  } catch (err) {

    console.log(err);

  }

};

1. *Fetch unlike*

const unlike = async (params, credentials, postId) => {

* *const unlike = async (params, credentials, postId) => {}:* Mendefinisikan fungsi *unlike* sebagai fungsi asinkron (dengan kata kunci *async*). Ini berarti fungsi ini bisa menggunakan await di dalamnya.
* *params, credentials, postId:* Parameter yang diterima oleh fungsi:
* *params*: Berisi data terkait pengguna, misalnya params.userId adalah ID pengguna yang menghapus "like".
* *credentials*: Berisi informasi autentikasi, misalnya credentials.t adalah token autentikasi.
* *postId*: ID dari postingan yang akan di-"*unlike*".

  try {

    let response = await fetch("/api/posts/unlike/", {

      method: "PUT",

      headers: {

        Accept: "application/json",

        "Content-Type": "application/json",

        Authorization: "Bearer " + credentials.t,

      },

      body: JSON.stringify({ userId: params.userId, postId: postId }),

    });

* *try { ... }:* Bagian ini mencoba menjalankan kode di dalamnya. Jika ada kesalahan (*error*) yang terjadi, kontrol akan berpindah ke blok *catch*.
* *let response = await fetch(...):* *fetch* digunakan untuk mengirim permintaan *HTTP*. *await* digunakan untuk menunggu respons dari *server* sebelum melanjutkan eksekusi kode berikutnya.
* *"/api/posts/unlike/": URL* *endpoint* di *server* yang akan menangani permintaan untuk menghapus "*like*" dari postingan.
* *method: "PUT":* Menggunakan metode *HTTP PUT* untuk memperbarui *data* di *server*.
* *headers: { ... }:* Menyertakan beberapa header dalam permintaan:
* *Accept: "application/json":* Memberitahu *server* bahwa klien mengharapkan respons dalam format *JSON*.
* "*Content-Type": "application/json":* Memberitahu server bahwa data yang dikirim dalam permintaan adalah *JSON*.
* *Authorization: "Bearer " + credentials.t*: Menyertakan token autentikasi dalam header *Authorization*.
* *body: JSON.stringify({ userId: params.userId, postId: postId }):* Menyertakan data dalam body permintaan sebagai *JSON*. *Data* ini berisi *userId* (*ID* pengguna yang menghapus "*like*") dan *postId* (*ID* dari postingan yang akan di-"*unlike*").

    return await response.json();

  } catch (err) {

    console.log(err);

  }

};

* *return await response.json():* Menunggu respons dari *server* dalam format *JSON* dan mengembalikannya dari fungsi unlike.
* *catch (err) { ... }:* Jika ada kesalahan yang terjadi dalam blok *try*, blok ini akan menangkap kesalahan tersebut.
* *console.log(err):* Mencetak kesalahan ke konsol untuk debugging.
* Tracking and counting likes on a post

Source code yang menggunakan metode *checkLike* pada *post*.*js* untuk mencari apabila pengguna yang sedang login direferensikan pada array post atau tidak :

const checkLike = (likes) => {

    let match = likes.indexOf(jwt.user.\_id) !== -1;

    return match;

  };

const [values, setValues] = useState({

    like: checkLike(props.post.likes),

    likes: props.post.likes.length,

    comments: props.post.comments,

  });

1. *checkLike*

const checkLike = (likes) => {

*const checkLike = (likes) => {}:* Mendefinisikan fungsi checkLike sebagai fungsi panah (*arrow function*) yang menerima satu parameter likes.

    let match = likes.indexOf(jwt.user.\_id) !== -1;

* *likes.indexOf(jwt.user.\_id) !== -1:* Mengecek apakah *ID* pengguna (*jwt.user.\_id*) ada di dalam *array* *likes*. *indexOf* mengembalikan indeks (posisi) dari elemen dalam *array*. Jika elemen tidak ditemukan, *indexOf* mengembalikan -1.
* *!== -1*: Membandingkan hasil *indexOf* dengan -1. Jika hasilnya tidak sama dengan -1, berarti *ID* pengguna ditemukan dalam *array* *likes*.
* *let match*: Menyimpan hasil pengecekan dalam variabel *match*.

    return match;

  };

*return match*: Mengembalikan nilai *match* (*boolean*) yang menunjukkan apakah pengguna telah "*like*" postingan tersebut (*true* jika ya, *false* jika tidak).

const [values, setValues] = useState({

*const [values, setValues] = useState(...):* Menggunakan *hook* *useState* untuk mendefinisikan *state* *values* dan fungsi *setValues* untuk memperbarui state tersebut. *useState* adalah *hook* yang memungkinkan penambahan *state* lokal pada komponen fungsional.

    like: checkLike(props.post.likes),

    likes: props.post.likes.length,

    comments: props.post.comments,

  });

* *like: checkLike(props.post.likes):* Menginisialisasi properti *like* dengan hasil dari fungsi *checkLike*, yang mengecek apakah pengguna telah "*like*" postingan ini.
* *likes: props.post.likes.length*: Menginisialisasi properti *likes* dengan jumlah "*like*" pada postingan, yang didapat dari panjang *array likes* (*props.post.likes.length*).
* *comments: props.post.comments*: Menginisialisasi properti *comments* dengan *array* *comments* dari postingan.

*Sourceh code* untuk menangani klik pada *icon* *like* dan *unlike*, gunakan metode *clickLike* untuk memanggil metode *fetch* yang sesuai (*like* atau *unlike* ketika diklik):

const clickLike = () => {

    let callApi = values.like ? unlike : like;

    callApi(

      {

        userId: jwt.user.\_id,

      },

      {

        t: jwt.token,

      },

      props.post.\_id

    ).then((data) => {

      if (data.error) {

        console.log(data.error);

      } else {

        setValues({ ...values, like: !values.like, likes: data.likes.length });

      }

    });

  };

1. *clickLike*

const clickLike = () => {

*clickLike* adalah sebuah fungsi yang didefinisikan menggunakan sintaks panah (*=>*).

    let callApi = values.like ? unlike : like;

Jika *values.like* bernilai *true* (artinya posting sudah disukai oleh pengguna), maka *callApi* akan diatur ke fungsi *unlike* (untuk menghapus suka). Jika *values.like* bernilai *false*, maka *callApi* akan diatur ke fungsi *like* (untuk menambah suka).

    callApi(

      {

        userId: jwt.user.\_id,

      },

      {

        t: jwt.token,

      },

      props.post.\_id

    ).then((data) => {

*callApi* (yang sekarang adalah *like* atau *unlike*) dipanggil dengan tiga argumen:

* Objek pertama berisi *userId*, yaitu *ID* pengguna.
* Objek kedua berisi *token (t)* yang digunakan untuk otentikasi.
* Argumen ketiga adalah *ID* *posting (props.post.\_id).*

Setelah *API* dipanggil, hasilnya (respon) akan ditangani oleh fungsi *.then*.

      if (data.error) {

        console.log(data.error);

      } else {

        setValues({ ...values, like: !values.like, likes: data.likes.length });

      }

    });

  };

Jika respon *API* mengandung *error* (*data.error*), *error* tersebut akan dicetak ke konsol. Jika tidak ada *error*, maka:

* Nilai *like* akan dibalik (dari *true* ke *false* atau sebaliknya).
* Jumlah *likes* diperbarui dengan panjang dari *array likes* yang diterima dari respon *API (data.likes.length).*
* Commenting

Source code untuk komponen comment :

<CardHeader

              avatar={

                <Avatar className={classes.smallAvatar} src={'/api/users/photo/'+auth.isAuthenticated().user.\_id}/>

              }

              title={ <TextField

                onKeyDown={addComment}

                multiline

                value={text}

                onChange={handleChange}

                placeholder="Write something ..."

                className={classes.commentField}

                margin="normal"

                />}

              className={classes.cardHeader}

        />

1. *Comment CardHeader*

<CardHeader

              avatar={

                <Avatar className={classes.smallAvatar} src={'/api/users/photo/'+auth.isAuthenticated().user.\_id}/>

              }

<*CardHeader* adalah komponen yang membentuk header dari sebuah kartu.

* *Avatar*: Menampilkan gambar profil pengguna.
* *className={classes.smallAvatar}:* Menambahkan kelas *CSS* untuk memberikan gaya khusus pada avatar.
* *src={'/api/users/photo/'+auth.isAuthenticated().user.\_id}:* Menentukan *URL* gambar avatar pengguna. *URL* ini diambil dari *server* dan menggabungkan *ID* pengguna *(auth.isAuthenticated().user.\_id)* untuk mendapatkan gambar yang benar.

              title={ <TextField

                onKeyDown={addComment}

                multiline

                value={text}

                onChange={handleChange}

                placeholder="Write something ..."

                className={classes.commentField}

                margin="normal"

                />}

* *TextField*: Ini adalah komponen input teks yang digunakan untuk menulis komentar.
* *onKeyDown={addComment}:* Menentukan fungsi *addComment* yang dipanggil ketika ada tombol ditekan di dalam bidang teks. Ini biasanya digunakan untuk menambahkan komentar ketika pengguna menekan tombol Enter.
* *multiline*: Membuat bidang teks menjadi *multiline*, sehingga pengguna bisa menulis komentar panjang.
* *value={text}:* Mengatur nilai bidang teks dengan variabel *text*, yang menyimpan teks komentar saat ini.
* *onChange={handleChange}:* Menentukan fungsi *handleChange* yang dipanggil setiap kali teks dalam bidang teks berubah. Ini digunakan untuk memperbarui nilai text.
* *placeholder="Write something ...":* Menampilkan teks *placeholder* di dalam bidang teks ketika belum ada teks yang ditulis.
* *className={classes.commentField}:* Menambahkan kelas *CSS* untuk memberikan gaya khusus pada bidang teks.
* *margin="normal":* Menambahkan margin normal di sekitar bidang teks untuk tata letak yang lebih baik.

              className={classes.cardHeader}

        />

Ditambahkan kelas *CSS* classes.*cardHeader* ke *CardHeader* untuk memberikan gaya khusus pada header kartu.

* Source code yang menyimpankan komen menggunakan metode addComment untuk memanggil metode fetch ketike kunci Enter dipencet:

1. *addComment*

const addComment = (event) => {

Ini adalah cara untuk mendefinisikan fungsi *addComment* menggunakan sintaks panah. Fungsi ini menerima satu *parameter*, yaitu *event*.

    if(event.keyCode == 13 && event.target.value){

Di dalam fungsi, kita memeriksa dua hal:

* Apakah tombol yang ditekan memiliki kode tombol 13 (*event.keyCode == 13*). Kode tombol 13 adalah tombol *Enter*.
* Apakah ada teks di dalam bidang teks (*event.target.value*).
* Jika kedua kondisi tersebut benar, maka kode di dalam blok *if* akan dijalankan.

      event.preventDefault()

*event.preventDefault()* mencegah tindakan default dari elemen tersebut. Misalnya, mencegah form dari pengiriman saat tombol *Enter* ditekan.

      comment({

        userId: jwt.user.\_id

      }, {

        t: jwt.token

      }, props.postId, {text: text}).then((data) => {

Fungsi *comment* dipanggil dengan empat argumen:

* Objek pertama: *{ userId: jwt.user.\_id }* - Ini adalah objek yang berisi *ID* pengguna (*userId*) yang diambil dari token *JWT*.
* Objek kedua: *{ t: jwt.token }* - Ini adalah objek yang berisi token (t) yang digunakan untuk otentikasi.
* Argumen ketiga: *props.postId* - Ini adalah ID dari *post* (*posting*) yang sedang dikomentari.
* Objek keempat: *{ text: text }* - Ini adalah objek yang berisi teks komentar (*text*).

Fungsi *comment* mengembalikan sebuah *Promise*, sehingga *.then* bisa digunakan untuk menangani hasilnya.

        if (data.error) {

          console.log(data.error)

        } else {

          setText('')

          props.updateComments(data.comments)

        }

      })

    }

  }

Di dalam *.then,* kita memeriksa apakah ada kesalahan (*data.error*):

* Jika ada kesalahan, kita mencetaknya ke konsol *(console.log(data.error)).*
* Jika tidak ada kesalahan:
* Kita mengosongkan bidang teks dengan *setText('').*
* Kita memperbarui daftar komentar dengan memanggil *props.updateComments(data.comments)* dan memberikan komentar terbaru (*data.comments*).

Membuat route PUT untuk update Post tersebut:

router.route('/api/posts/comment')

  .put(authCtrl.requireSignin, postCtrl.comment)

router.route('/api/posts/comment')

Kode ini mendefinisikan rute untuk *URL /api/posts/comment*. Artinya, ketika ada permintaan (*request*) ke *URL* ini, rute ini akan diproses.

  .put(authCtrl.requireSignin, postCtrl.comment)

* *PUT*: Menentukan bahwa rute ini menggunakan metode *HTTP PUT*. Artinya, permintaan ke *URL* ini harus menggunakan metode *PUT*.
* *authCtrl.requireSignin*: Middleware ini akan dijalankan terlebih dahulu. Middleware ini memeriksa apakah pengguna sudah masuk. Jika pengguna belum masuk, permintaan akan dihentikan dan pengguna mungkin mendapatkan respon kesalahan.
* *postCtrl.comment*: Jika *middleware authCtrl.requireSignin* berhasil (artinya pengguna sudah masuk), maka fungsi *postCtrl*.comment akan dijalankan. Fungsi ini menangani logika untuk menambah komentar ke posting yang relevan.

Source code untuk mencari post relevan dan mengupdate post tersebut, didasarkan pada ID-nya dengan menggunakan metode *comment controller* pada file *post.controller.js :*

const comment = async (req, res) => {

  let comment = req.body.comment

  comment.postedBy = req.body.userId

  try{

    let result = await Post.findByIdAndUpdate(req.body.postId, {$push: {comments: comment}}, {new: true})

                            .populate('comments.postedBy', '\_id name')

                            .populate('postedBy', '\_id name')

                            .exec()

    res.json(result)

  }catch(err){

    return res.status(400).json({

      error: errorHandler.getErrorMessage(err)

    })

  }

}

1. *Comment* *Controller*

const comment = async (req, res) => {

Ini adalah fungsi asinkron yang menangani permintaan untuk menambahkan komentar. Fungsi ini menerima dua parameter:

* *req*: Objek permintaan (*request*) yang berisi data dari klien.
* *res*: Objek respon (*response*) yang digunakan untuk mengirim balik data ke klien.

  let comment = req.body.comment

  comment.postedBy = req.body.userId

* *req.body.comment*: Mengambil data komentar dari body permintaan.
* *comment.postedBy = req.body.userId*: Menambahkan properti *postedBy* ke objek komentar, yang berisi *ID* pengguna yang membuat komentar.

  try{

    let result = await Post.findByIdAndUpdate(req.body.postId, {$push: {comments: comment}}, {new: true})

                            .populate('comments.postedBy', '\_id name')

                            .populate('postedBy', '\_id name')

                            .exec()

    res.json(result)

Blok *try* digunakan untuk mencoba menjalankan kode yang mungkin menghasilkan kesalahan. Jika ada kesalahan, blok *catch* akan menangkapnya dan mengirim respon *error*.

* *Post.findByIdAndUpdate*: Fungsi ini mencari posting berdasarkan ID (*req.body.postId*) dan memperbaruinya.
* *{ $push: { comments: comment } }:* Menambahkan (*push*) komentar baru ke array komentar dalam posting.
* *{ new: true }:* Menginstruksikan *MongoDB* untuk mengembalikan versi terbaru dari dokumen yang diperbarui.
* *.populate('comments.postedBy', '\_id name'):* Mengisi (*populate*) informasi tentang pengguna yang membuat komentar dengan hanya mengambil *\_id* dan *name*.
* .*populate('postedBy', '\_id name'):* Mengisi informasi tentang pengguna yang membuat posting.
* *.exec():* Menjalankan *query* dan mengembalikan hasilnya.
* *res.json(result):* Jika operasi berhasil, mengirim hasil (*posting* dengan komentar baru) sebagai respon dalam format *JSON*.

  }catch(err){

    return res.status(400).json({

      error: errorHandler.getErrorMessage(err)

    })

  }

}

Jika terjadi kesalahan, mengirim respon dengan *status* *400 (Bad Request)* dan pesan kesalahan dalam format *JSON*. *errorHandler.getErrorMessage(err)* digunakan untuk mendapatkan pesan kesalahan yang lebih mudah dipahami.

Source code untuk metode *fetch* yang mengambil ID current user, ID post, dan comment objek dari *view*, *source code* ini terletak pada api-post.js:

const comment = async (params, credentials, postId, comment) => {

  try {

    let response = await fetch("/api/posts/comment/", {

      method: "PUT",

      headers: {

        Accept: "application/json",

        "Content-Type": "application/json",

        Authorization: "Bearer " + credentials.t,

      },

      body: JSON.stringify({

        userId: params.userId,

        postId: postId,

        comment: comment,

      }),

    });

    return await response.json();

  } catch (err) {

    console.log(err);

  }

};

1. *Fetch* *Comment*

const comment = async (params, credentials, postId, comment) => {

*comment* adalah fungsi asinkron yang berarti ia bisa menggunakan await di dalamnya. Fungsi ini menerima empat parameter:

* *params*: Berisi *parameter*, dalam hal ini kita butuh *userId*.
* *credentials*: Berisi informasi kredensial pengguna, khususnya token (*credentials.t*).
* *postId*: *ID* dari *posting* yang ingin diberi komentar.
* *comment*: Objek komentar yang ingin ditambahkan.

  try {

Blok *try* digunakan untuk mencoba menjalankan kode yang mungkin menghasilkan kesalahan. Jika ada kesalahan, blok *catch* akan menangkapnya dan mencetak kesalahan tersebut ke konsol.

    let response = await fetch("/api/posts/comment/", {

      method: "PUT",

      headers: {

        Accept: "application/json",

        "Content-Type": "application/json",

        Authorization: "Bearer " + credentials.t,

      },

      body: JSON.stringify({

        userId: params.userId,

        postId: postId,

        comment: comment,

      }),

    });

* *fetch*: Fungsi ini digunakan untuk mengirim permintaan *HTTP*.
* *URL: "/api/posts/comment/"* adalah *URL* *endpoint* di *server* yang akan menerima permintaan untuk menambahkan komentar.
* *method: "PUT"* menunjukkan bahwa kita memperbarui *data* (menambahkan komentar ke posting yang sudah ada).
* *headers*: Berisi informasi tentang jenis konten yang dikirim dan otorisasi:
* *Accept: "application/json":* Mengindikasikan bahwa kita mengharapkan respon dalam format JSON.
* *"Content-Type": "application/json":* Menunjukkan bahwa data yang dikirim dalam format *JSON*.
* *Authorization: "Bearer " + credentials.t*: Mengirim token untuk otentikasi.
* body: Berisi data yang akan dikirim dalam format *JSON*. Data ini mencakup *userId, postId,* dan *comment*.

    return await response.json();

Setelah permintaan fetch selesai, kita mengonversi respon ke format JSON dan mengembalikannya.

  } catch (err) {

    console.log(err);

  }

};

Jika ada kesalahan selama proses permintaan, kesalahan tersebut akan ditangkap dan dicetak ke konsol.

* Deleting comment

Source code untuk membuat API uncomment pada route PUT:

router.route('/api/posts/uncomment')

  .put(authCtrl.requireSignin, postCtrl.uncomment)

1. API delete comment

router.route('/api/posts/uncomment')

Ini mendefinisikan sebuah rute (*route*) untuk *URL /api/posts/uncomment.* Artinya, ketika ada permintaan ke *URL* ini, rute ini akan diproses.

  .put(authCtrl.requireSignin, postCtrl.uncomment)

* *PUT: Metode HTTP PUT* digunakan untuk memperbarui sumber daya yang ada. Dalam konteks ini, digunakan untuk menghapus komentar dari sebuah posting.
* *Middleware* *authCtrl.requireSignin* adalah fungsi yang dijalankan sebelum permintaan utama diproses. Fungsi ini memeriksa apakah pengguna sudah masuk (*signed in*). Jika belum, permintaan akan ditolak.
* *Controller postCtrl.uncomment*: Jika pengguna sudah masuk, fungsi ini akan dijalankan. Fungsi ini mengandung logika untuk menghapus komentar dari posting yang ditentukan.

Metode uncomment ini akan mencari post yang relevan berdasarkan ID-nya dan mengeluarkan (pull) comment itu menggunakan ID komen tersebut, metode ini diimplementasikan pada file post-controller.js***:***

const uncomment = async (req, res) => {

  let comment = req.body.comment

  try{

    let result = await Post.findByIdAndUpdate(req.body.postId, {$pull: {comments: {\_id: comment.\_id}}}, {new: true})

                          .populate('comments.postedBy', '\_id name')

                          .populate('postedBy', '\_id name')

                          .exec()

    res.json(result)

  }catch(err){

    return res.status(400).json({

      error: errorHandler.getErrorMessage(err)

    })

  }

}

1. Uncomment method

const uncomment = async (req, res) => {

*uncomment* adalah fungsi asinkron yang berarti ia bisa menggunakan *await* di dalamnya. Fungsi ini menerima dua parameter:

* *req*: Objek permintaan (*request*) yang berisi data dari klien.
* *res*: Objek respon (*response*) yang digunakan untuk mengirim balik data ke klien.

  let comment = req.body.comment

*req.body.comment* mengandung objek komentar yang akan dihapus. Objek ini diambil dari body permintaan.

  try{

Blok *try* digunakan untuk mencoba menjalankan kode yang mungkin menghasilkan kesalahan. Jika ada kesalahan, blok *catch* akan menangkapnya dan mengirim respon error.

    let result = await Post.findByIdAndUpdate(req.body.postId, {$pull: {comments: {\_id: comment.\_id}}}, {new: true})

                          .populate('comments.postedBy', '\_id name')

                          .populate('postedBy', '\_id name')

                          .exec()

* *Post.findByIdAndUpdate*: Fungsi ini mencari posting berdasarkan *ID* *(req.body.postId)* dan memperbaruinya.
* *{ $pull: { comments: { \_id: comment.\_id } } }:* Menggunakan operator *$pull* untuk menghapus komentar yang memiliki *ID* yang sesuai *(comment.\_id)* dari *array* komentar dalam posting.
* *{ new: true }:* Menginstruksikan *MongoDB* untuk mengembalikan versi terbaru dari dokumen yang diperbarui.
* *.populate('comments.postedBy', '\_id name'):* Mengisi (*populate*) informasi tentang pengguna yang membuat komentar dengan hanya mengambil *\_id* dan *name*.
* .*populate('postedBy', '\_id name'):* Mengisi informasi tentang pengguna yang membuat posting.
* *.exec():* Menjalankan *query* dan mengembalikan hasilnya.

    res.json(result)

Jika operasi berhasil, mengirim hasil (posting dengan komentar yang sudah dihapus) sebagai respon dalam format *JSON*.

  }catch(err){

    return res.status(400).json({

      error: errorHandler.getErrorMessage(err)

    })

  }

}

Jika terjadi kesalahan, mengirim respon dengan *status 400 (Bad Request)* dan pesan kesalahan dalam format JSON. *errorHandler.getErrorMessage(err)* digunakan untuk mendapatkan pesan kesalahan yang lebih mudah dipahami.

Buat source code metode fetch untuk menghapus komen pada api-post.js :

const uncomment = async (params, credentials, postId, comment) => {

  try {

    let response = await fetch("/api/posts/uncomment/", {

      method: "PUT",

      headers: {

        Accept: "application/json",

        "Content-Type": "application/json",

        Authorization: "Bearer " + credentials.t,

      },

      body: JSON.stringify({

        userId: params.userId,

        postId: postId,

        comment: comment,

      }),

    });

    return await response.json();

  } catch (err) {

    console.log(err);

  }

};

1. Fetch uncomment

const uncomment = async (params, credentials, postId, comment) => {

*uncomment* adalah fungsi asinkron yang berarti ia bisa menggunakan await di dalamnya. Fungsi ini menerima empat parameter:

* *params*: Berisi parameter yang diperlukan, seperti *userId*.
* *credentials*: Berisi informasi kredensial pengguna, khususnya token (*credentials.t*).
* *postId*: *ID* dari posting yang ingin dihapus komentarnya.
* *comment*: Objek komentar yang ingin dihapus.

  try {

Blok *try* digunakan untuk mencoba menjalankan kode yang mungkin menghasilkan kesalahan. Jika ada kesalahan, blok *catch* akan menangkapnya dan mencetak kesalahan tersebut ke konsol.

    let response = await fetch("/api/posts/uncomment/", {

      method: "PUT",

      headers: {

        Accept: "application/json",

        "Content-Type": "application/json",

        Authorization: "Bearer " + credentials.t,

      },

      body: JSON.stringify({

        userId: params.userId,

        postId: postId,

        comment: comment,

      }),

    });

* *fetch*: Fungsi ini digunakan untuk mengirim permintaan *HTTP*.
* *URL: "/api/posts/uncomment/"* adalah *URL* *endpoint* di *server* yang akan menerima permintaan untuk menghapus komentar.
* *method: "PUT"* menunjukkan bahwa kita memperbarui data (menghapus komentar dari posting yang sudah ada).
* *headers*: Berisi informasi tentang jenis konten yang dikirim dan otorisasi:
* *Accept: "application/json":* Mengindikasikan bahwa kita mengharapkan respon dalam format JSON.
* "*Content-Type": "application/json*": Menunjukkan bahwa data yang dikirim dalam format *JSON*.
* *Authorization: "Bearer* " *+ credentials.t*: Mengirim token untuk otentikasi.
* *body*: Berisi data yang akan dikirim dalam format *JSON*. Data ini mencakup *userId, postId*, dan *comment*.

    return await response.json();

Setelah permintaan fetch selesai, kita mengonversi respon ke format JSON dan mengembalikannya.

  } catch (err) {

    console.log(err);

  }

};

Jika ada kesalahan selama proses permintaan, kesalahan tersebut akan ditangkap dan dicetak ke konsol.

Source code untuk metode deleteComment untuk *fetch* API uncomment dan update comments :

const deleteComment = (comment) => (event) => {

    uncomment(

      {

        userId: jwt.user.\_id,

      },

      {

        t: jwt.token,

      },

      props.postId,

      comment

    ).then((data) => {

      if (data.error) {

        console.log(data.error);

      } else {

        props.updateComments(data.comments);

      }

    });

  };

1. Metode *deleteComment*

const deleteComment = (comment) => (event) => {

*deleteComment* adalah sebuah fungsi yang menerima satu *parameter*, yaitu *comment*. Fungsi ini mengembalikan fungsi lain yang menerima parameter event. Ini disebut fungsi pengurutan atau *closure*, yang memungkinkan kita untuk menyimpan nilai comment di dalamnya.

    uncomment(

      {

        userId: jwt.user.\_id,

      },

      {

        t: jwt.token,

      },

      props.postId,

      comment

)

*uncomment*: Ini adalah fungsi yang mengirim permintaan ke *server* untuk menghapus komentar. *Parameter* untuk *uncomment*:

* *{ userId: jwt.user.\_id }:* Mengirim ID pengguna yang ingin menghapus komentar.
* *{ t: jwt.token }:* Mengirim token otentikasi pengguna.
* *props.postId*: Mengirim *ID posting* dari mana komentar akan dihapus.
* *comment*: Mengirim objek komentar yang akan dihapus.

.then((data) => {

      if (data.error) {

        console.log(data.error);

      } else {

        props.updateComments(data.comments);

      }

    });

  };

*.then((data) => { ... }):* Setelah permintaan *uncomment* selesai, kita memproses respon.

* Jika Ada Kesalahan: *if (data.error) { console.log(data.error); }:* Jika respon mengandung kesalahan (data.error), kesalahan tersebut dicetak ke konsol.
* Jika Berhasil: *else { props.updateComments(data.comments); }:* Jika tidak ada kesalahan, kita memanggil props.updateComments dengan daftar komentar terbaru (data.comments) untuk memperbarui tampilan komentar di antarmuka pengguna.

Fungsi-fungsi tersebut menggunakan fetch untuk mengirim permintaan ke endpoint yang sesuai pada server, menggunakan metode yang sesuai (POST, GET, PUT, DELETE) dan menyertakan header yang diperlukan. Jika permintaan berhasil, respons dari server diurai sebagai JSON dan dikembalikan dari fungsi. Jika terjadi kesalahan selama proses, kesalahan ditangkap dan dicetak ke konsol.

Jadi, secara keseluruhan, komponen Post ini bertanggung jawab untuk menampilkan satu postingan dan memberikan fungsionalitas seperti menyukai postingan, menghapus postingan, dan menampilkan komentar.

* **Who to Follow**
* Eye button (Look at other people’s account)

Source code yang membiarkan pengguna melihat profile pengguna lainnya dengan mengklik icon mata*:*

<Link to={"/user/" + item.\_id}>

                      <IconButton

                        variant="contained"

                        color="secondary"

                        className={classes.viewButton}

                      >

                        <ViewIcon />

                      </IconButton>

                    </Link>

1. Eye icon

<Link to={"/user/" + item.\_id}>

*Link* adalah komponen dari *react-router-dom* yang digunakan untuk membuat tautan di aplikasi *React*. Ketika pengguna mengklik tautan ini, mereka akan diarahkan ke halaman profil pengguna dengan ID item.\_id.

* *to* adalah properti yang menentukan ke mana tautan akan mengarahkan pengguna.
* *"/user/" + item.\_id* adalah cara untuk membuat URL dinamis. item.\_id adalah ID dari pengguna yang saat ini sedang ditampilkan.

                      <IconButton

                        variant="contained"

                        color="secondary"

                        className={classes.viewButton}

                      >

                        <ViewIcon />

                      </IconButton>

                    </Link>

* *IconButton* adalah komponen dari *Material-UI* yang digunakan untuk membuat tombol dengan ikon.
* *variant="contained":* Menentukan gaya tombol. Meskipun ini tidak biasanya diterapkan pada *IconButton*, biasanya digunakan pada *Button*. Mungkin tidak diperlukan di sini.
* *color="secondary":* Menentukan warna tombol. *Material-UI* menyediakan beberapa pilihan warna seperti *primary* dan *secondary*.
* *className={classes.viewButton}:* Menentukan kelas CSS untuk tombol. *classes.viewButton* adalah gaya yang didefinisikan menggunakan *makeStyles* atau cara lain dalam *Material-UI*.
* *<ViewIcon />:* Ini adalah ikon yang akan ditampilkan di dalam tombol. *ViewIcon* adalah komponen ikon dari *Material-UI* atau ikon yang diimpor.
* Follow button

**Source code:**

                    <Button

                      aria-label="Follow"

                      variant="contained"

                      color="primary"

                      onClick={() => {

                        clickFollow(item, i);

                      }}

                    >

                      Follow

                    </Button>

* *<Button>:* Ini adalah komponen dari *Material-UI* yang digunakan untuk membuat tombol.
* *aria-label="Follow":* Ini adalah atribut untuk aksesibilitas. Ini memberikan deskripsi untuk pembaca layar tentang fungsi tombol, yang dalam kasus ini adalah "*Follow*".
* variant*="contained":* Menentukan gaya tombol. contained berarti tombol ini akan terlihat seperti tombol yang terangkat dengan latar belakang warna solid.
* *color="primary":* Menentukan warna tombol. *Material-UI* menyediakan beberapa pilihan warna seperti *primary* dan *secondary*. primary biasanya biru.
* *onClick={() => { clickFollow(item, i); }}:* Fungsi ini akan dipanggil ketika tombol diklik. Dalam hal ini, fungsi *clickFollow* dipanggil dengan dua argumen: *item* dan *i.*

Source code untuk metode *clickFollow* yang mengimplementasi kemampuan program untuk mengupdate *list* pengguna *Who to Follow*:

const clickFollow = (user, index) => {

    follow(

      {

        userId: jwt.user.\_id,

      },

      {

        t: jwt.token,

      },

      user.\_id

    ).then((data) => {

      if (data.error) {

        console.log(data.error);

      } else {

        let toFollow = values.users;

        toFollow.splice(index, 1);

        setValues({

          ...values,

          users: toFollow,

          open: true,

          followMessage: `Following ${user.name}!`,

        });

      }

    });

  };

1. *clickFollow*

const clickFollow = (user, index) => {

*const clickFollow = (user, index) => { ... }:* Mendefinisikan fungsi clickFollow menggunakan sintaks panah (*arrow function*).

* *user*: Argumen pertama yang mewakili pengguna yang akan diikuti.
* *index*: Argumen kedua yang mewakili posisi (indeks) pengguna dalam daftar.

    follow(

      {

        userId: jwt.user.\_id,

      },

      {

        t: jwt.token,

      },

      user.\_id

    )

* *follow* adalah fungsi yang dipanggil dengan tiga argumen:
* Objek pengguna*: { userId: jwt.user.\_id }* di mana *jwt.user.\_id* adalah *ID* dari pengguna yang sedang login.
* Objek *token: { t: jwt.token }* di mana jwt.token adalah token autentikasi pengguna yang sedang login.
* *ID* pengguna yang akan diikuti: *user.\_id* di mana *user.\_id* adalah *ID* dari pengguna yang akan diikuti.

.then((data) => {

      if (data.error) {

        console.log(data.error);

      } else {

        let toFollow = values.users;

        toFollow.splice(index, 1);

        setValues({

          ...values,

          users: toFollow,

          open: true,

          followMessage: `Following ${user.name}!`,

        });

      }

    });

  };

* *follow(...).then((data) => { ... }):* Setelah follow selesai, kita menunggu hasilnya dengan .then.
* *data*: Objek hasil yang diterima dari fungsi follow.

      if (data.error) {

        console.log(data.error);

}

Jika *data.error* ada, itu berarti ada kesalahan dan kita mencetak pesan kesalahan ke konsol dengan *console.log(data.error).*

else {

        let toFollow = values.users;

        toFollow.splice(index, 1);

        setValues({

          ...values,

          users: toFollow,

          open: true,

          followMessage: `Following ${user.name}!`,

        });

      }

    });

  };

* *let toFollow = values.users;:* Salin daftar pengguna yang ada di values.users.
* *toFollow.splice(index, 1);:* Hapus pengguna dari daftar toFollow pada posisi index.
* *setValues({ ...values, users: toFollow, open: true, followMessage: Following ${user.name}!, });:* Memperbarui *state* dengan nilai baru.
* *...values:* Menyalin semua nilai yang ada dalam *values* saat ini.
* *users: toFollow:* Memperbarui daftar pengguna menjadi *toFollow* yang baru.
* *open: true:* Menyetel open ke true, mungkin untuk menampilkan pesan atau notifikasi.
* *followMessage: Following ${user.name}!``:* Menyetel pesan follow dengan nama pengguna yang baru diikuti.

Source code yang menunjukkan notifikasi sementara bahwa pengguna sukses follow pengguna lainnya:

<Snackbar

          anchorOrigin={{

            vertical: 'bottom',

            horizontal: 'right',

          }}

          open={values.open}

          onClose={handleRequestClose}

          autoHideDuration={6000}

          message={<span className={classes.snack}>{values.followMessage}</span>}

      />

1. Property SnackBar

<Snackbar

          anchorOrigin={{

            vertical: 'bottom',

            horizontal: 'right',

          }}

*anchorOrigin* menentukan posisi di mana *Snackbar* akan muncul di layar.

* *vertical: 'bottom'*: Posisinya di bagian bawah layar.
* *horizontal: 'right':* Posisinya di sisi kanan layar.

          open={values.open}

Menentukan apakah *Snackbar* akan ditampilkan atau tidak. *values.open* adalah *boolean* (*true* atau *false*). Jika *true*, Snackbar akan muncul; jika *false*, Snackbar tidak akan muncul.

          onClose={handleRequestClose}

Fungsi yang akan dipanggil ketika *Snackbar* ditutup. *handleRequestClose* adalah fungsi yang mendefinisikan apa yang harus dilakukan saat *Snackbar* ditutup (misalnya, memperbarui *state* untuk menyembunyikan *Snackbar*).

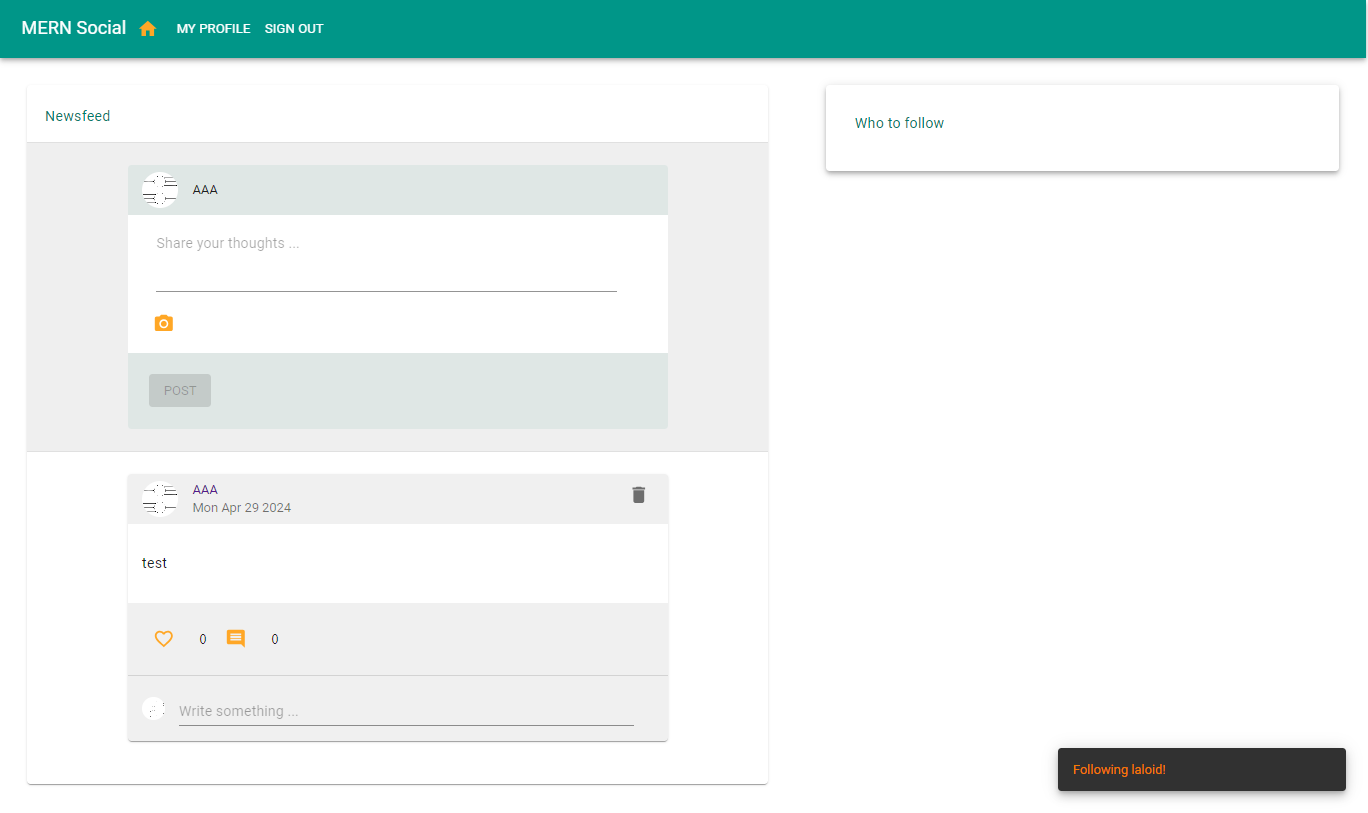
          autoHideDuration={6000}

Menentukan berapa lama *Snackbar* akan ditampilkan sebelum secara otomatis disembunyikan. 6000 berarti *Snackbar* akan terlihat selama 6000 milidetik (6 detik) sebelum menghilang.

          message={<span className={classes.snack}>{values.followMessage}</span>}

      />

Pesan yang akan ditampilkan di dalam Snackbar. *<span className={classes.snack}>{values.followMessage}</span>* menampilkan pesan *values.followMessage* di dalam elemen *<span>* dengan kelas CSS classes.snack.

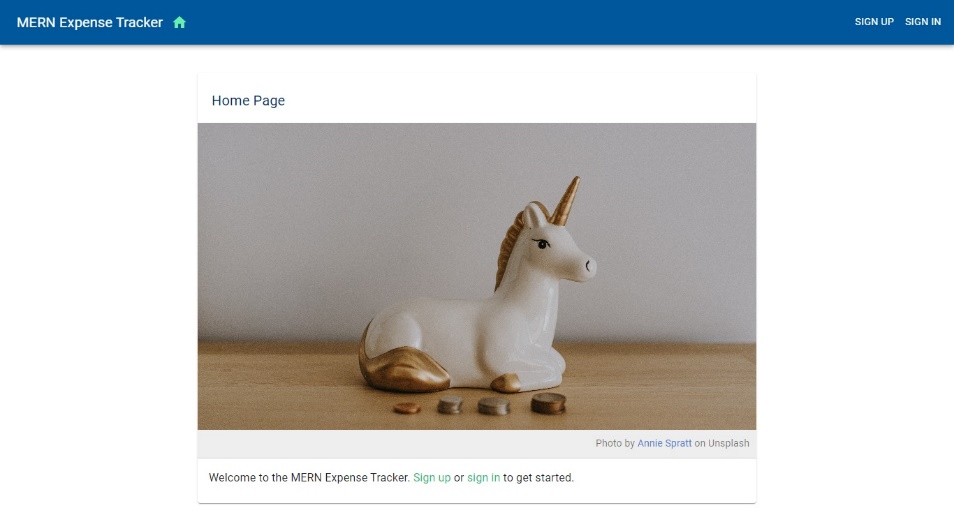
****

# 3.2 Aplikasi Expenses Tracker

## Hasil Aplikasi

Berikut ini merupkan tampilan-tampilan pada aplikasi Tracker Expenses.

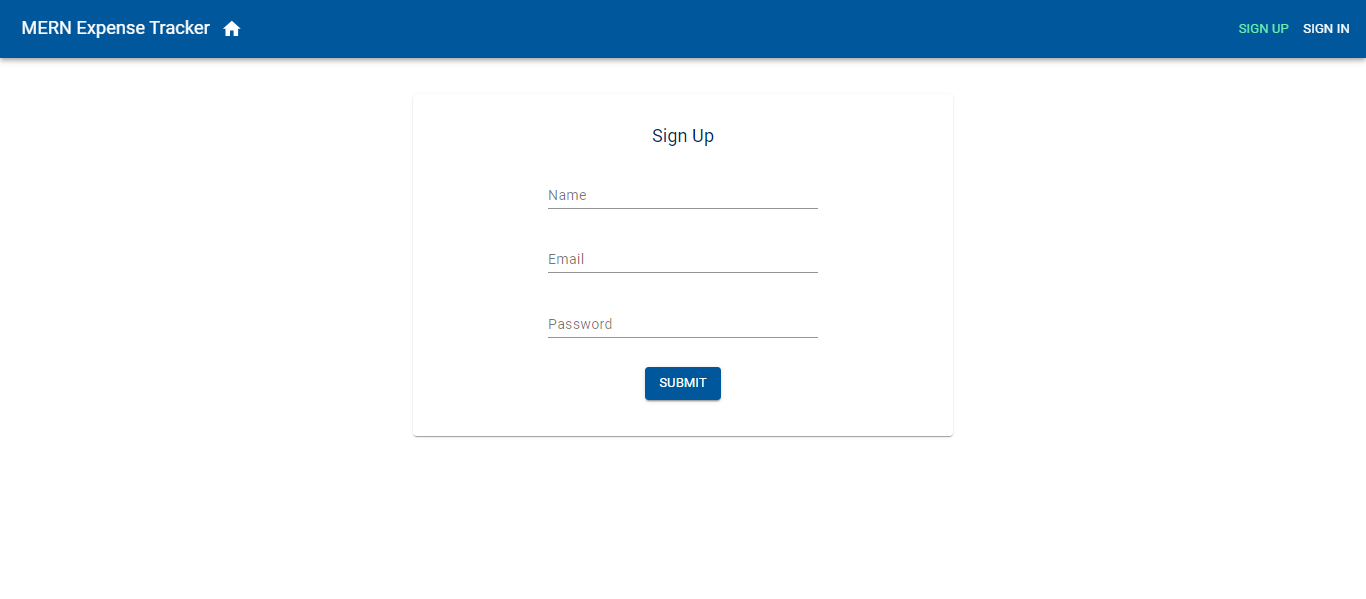
### 3.2.1.1 Halaman Utama



Gambar 2.31 Halaman Utama Expense Tracker

Berikut adalah tampilan halaman utama Expense Tracker.Dalam desain UI/UX untuk aplikasi expense tracker, halaman utama memainkan peran krusial sebagai titik awal interaksi pengguna dengan aplikasi. Halaman utama Expense Tracker terdapat *Sign Up* dan *Sign In.*

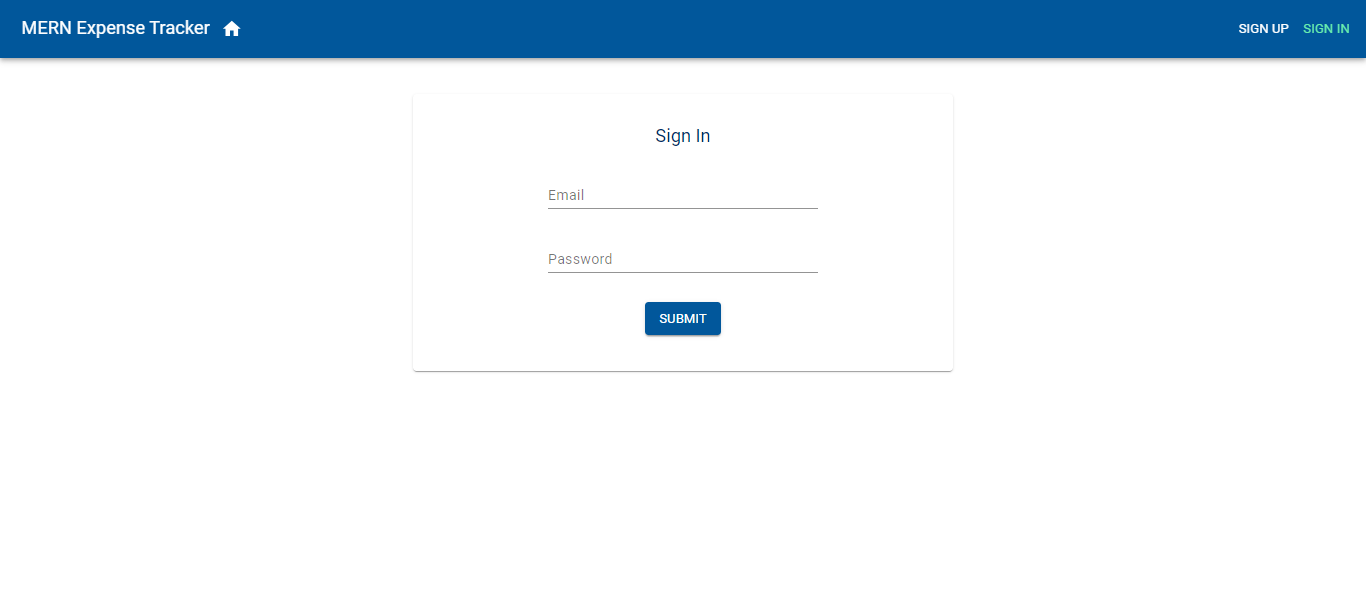
### 3.2.1.2 Halaman Sign Up



Gambar 2.32 Halaman Sign Up Expense Tracler

Kemudian memasuki pada bagian Halaman *Sign Up*. jika pengguna belum pernah menggunakan aplikasi ini maka bisa melakukan *Sign Up*. Daftarkan Nama, Email, serta Kata Sandi.

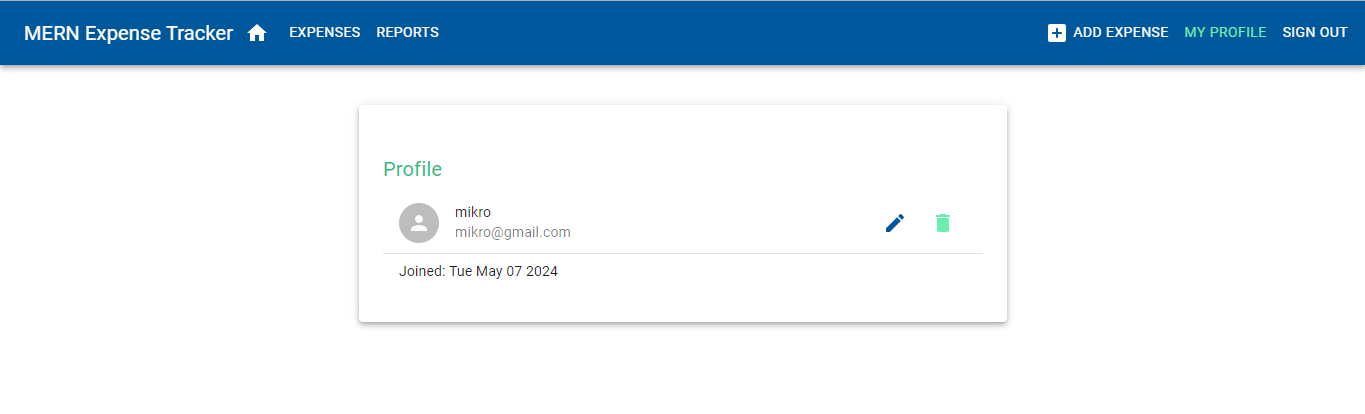
### 3.2.1.3 Halaman Sign In



Gambar 2.32 Halaman Sign In Expense Tracker

Berikut merupakan tampilan halaman Sign In. Setelah melakukan pendaftaran akun/sign up, maka akan memasuki halaman Sign In. Pada bagian ini akan memasukan email dan kata sandi yang sudah di daftarkan. Fungsi sign in dalam konteks aplikasi atau layanan online adalah untuk memungkinkan pengguna yang sudah terdaftar mengakses akun mereka dan menggunakan fitur serta layanan yang tersedia.

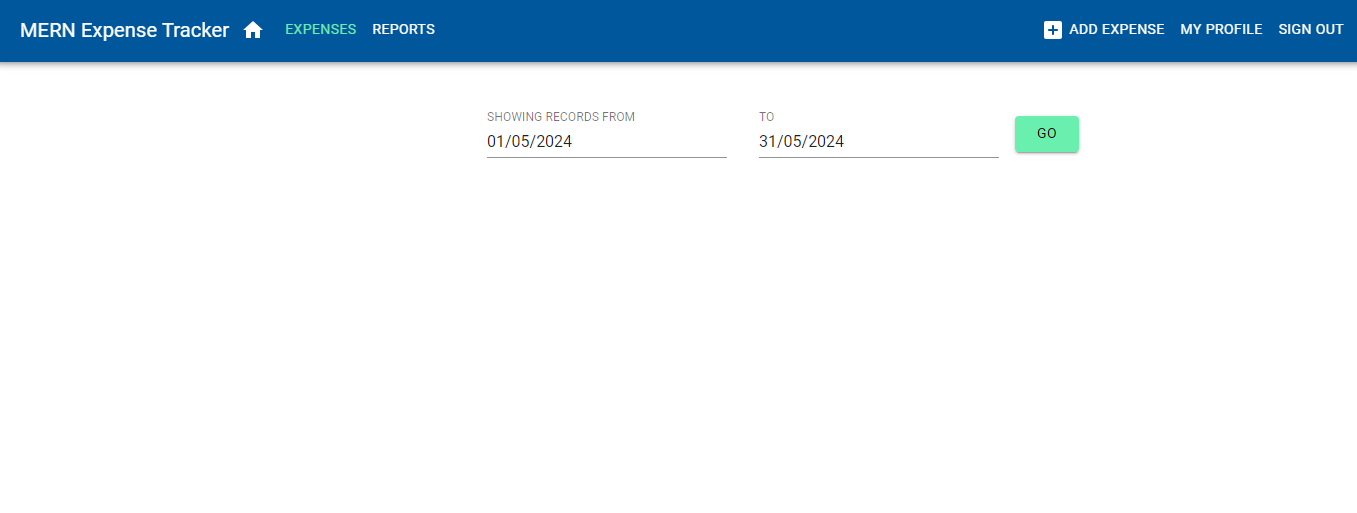
### 3.2.1.4 Halaman My Profile



Gambar 2.33 Halaman My Profile Expense Tracker

Memasuki Halaman *My Profile*, terdapat fitur *Expenses, Reports, Add Expense, Sign Out*. Halaman tersebut akan menampilkan hari, tanggal, dan tahun sejak pembuatan akun *Expense Tracker.*

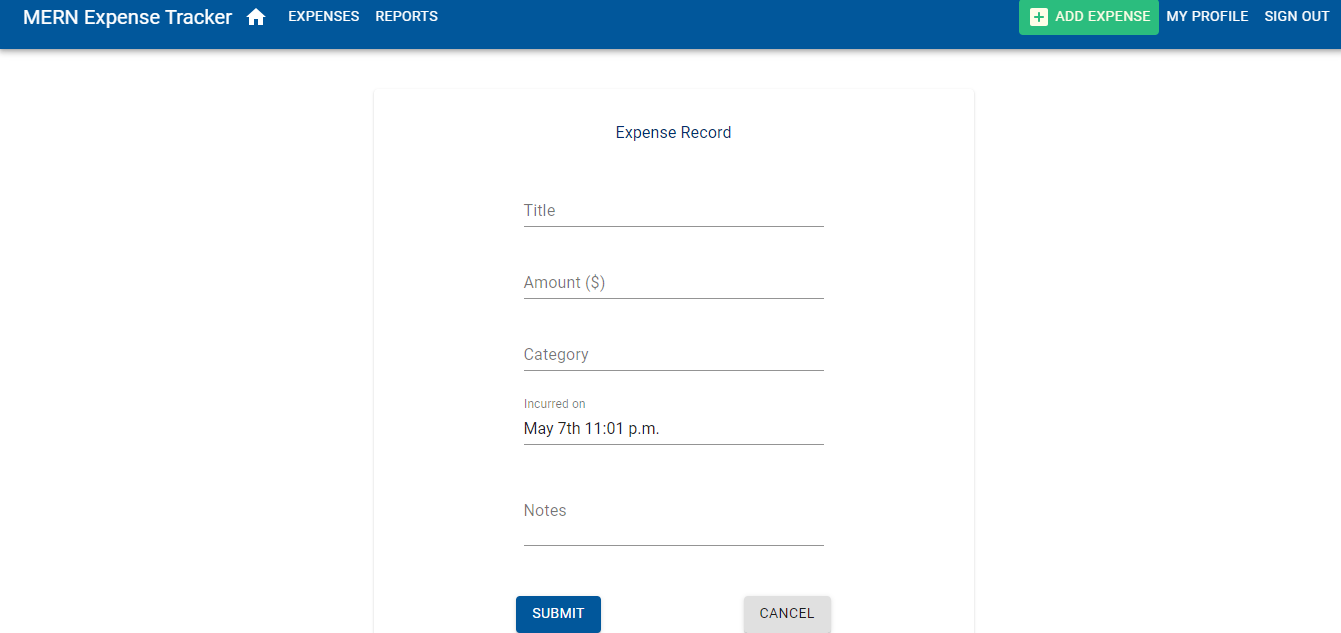
### 3.2.1.5 Halaman Expenses



Gambar 2.34 Halaman Expense Tracker

Berikut ini merupuakan Halaman *Expenses* dapat memfilter record file yang telah kita tambahkan dari tanggal dan tahun kapan setelah dibuat.

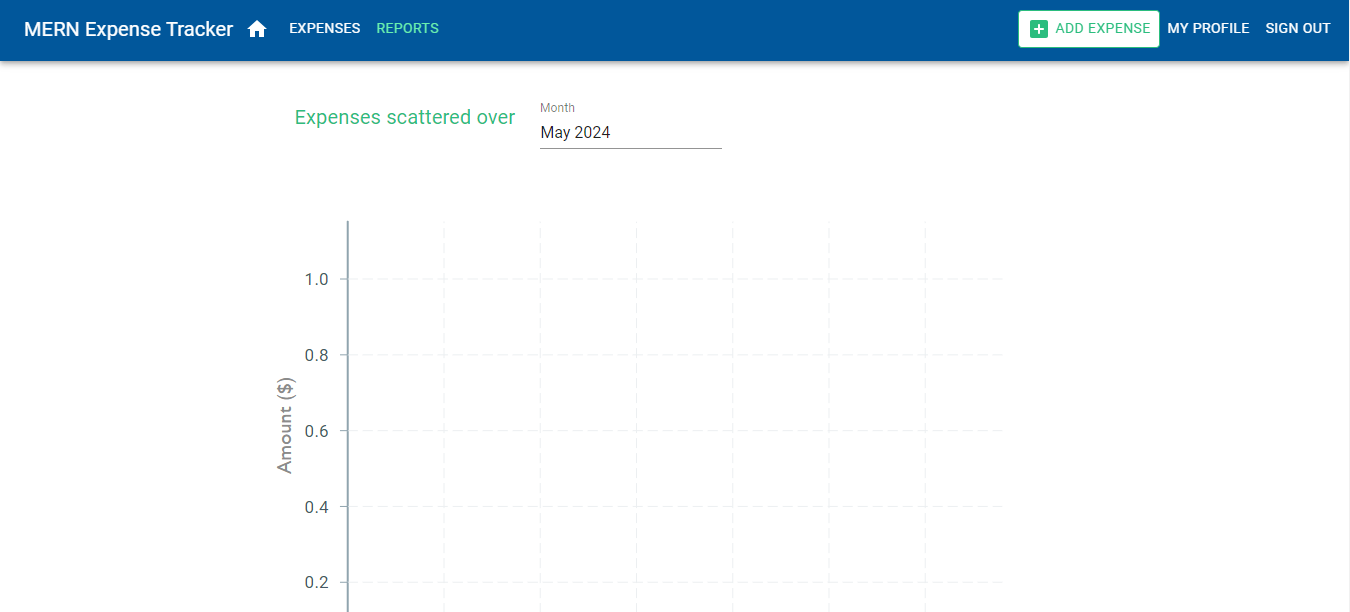
### 3.2.1.6 Halaman Add Expenses



Gambar 2.35 Halaman Add Expense

Pada tampilan ini merupakan Halaman Add Expenses. Terdapat judul, jumlah, kategori, bulan, tanggal, dan waktu terjadi, dan catatan.

### 3.2.1.1 Halaman Reports Menu



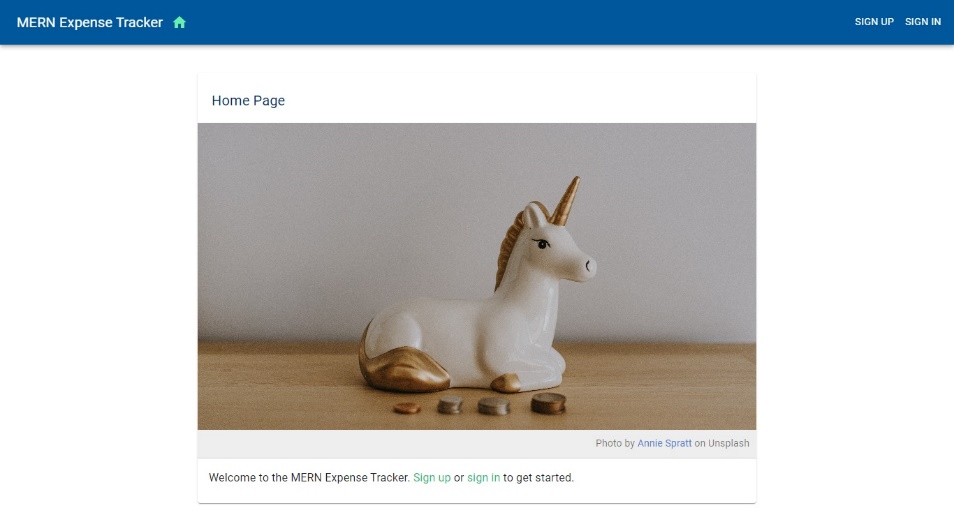
Gambar 2.36 Halaman Reports Menu

Pada bagian Halaman Reports menu, terdapat expenses, reports, add expenses, my profile, dan sign out.

## 3.2.2 Pembahasan

## 3.2.2.1 UI/UX

#### A. Halaman Utama

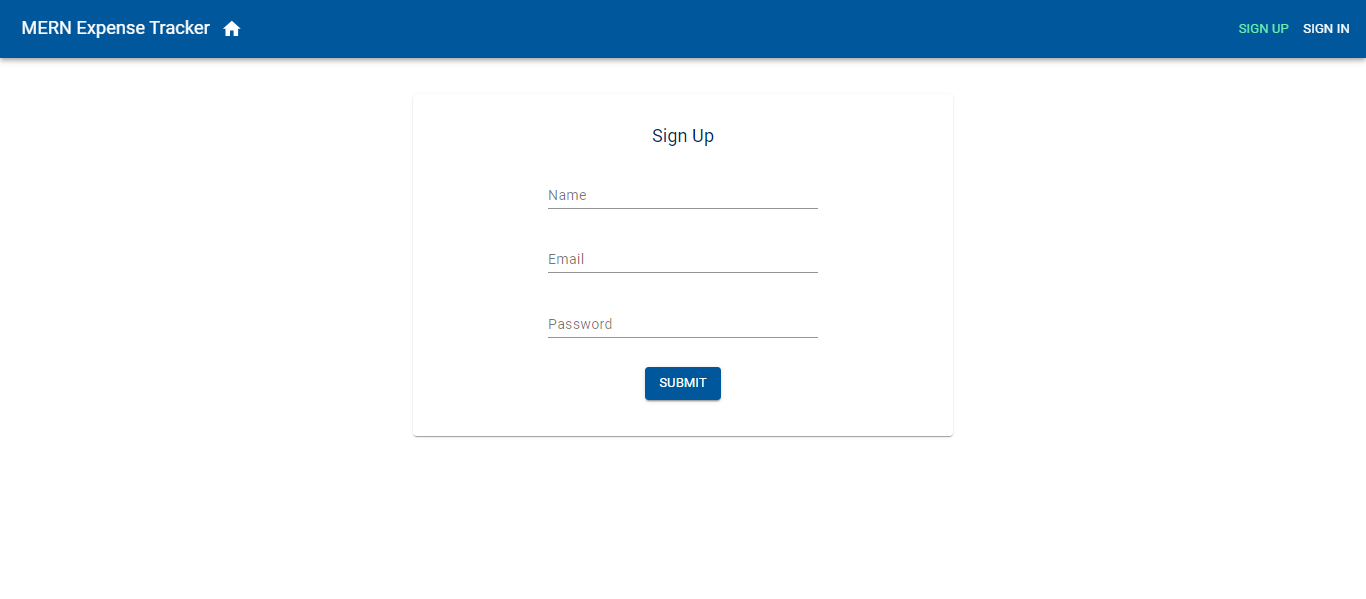


Gambar 2.37 Halaman Utama

Gambar berikut ini adalah halaman Utama yang akan muncul pertama kali pada saat membuka aplikasi Expense Tracker sebelum memasukan akun kita. Halaman ini menampilkan fituer sign up dan sign in, yang dimana pengguna dapat memasukan akun dan menggunakan aplikasi Expense Tracker tersebut.

Pada teks "Welcome to the MERN Expense Tracker. Sign Up or Sign In to get started."terletak ditengah-tengah halaman yang berfungsi untuk menyambut pengguna. Dan untuk teks "Sign Up or Sign In to get started" berfungsi untuk masuk atau membuat sebuah akun yang dimana agar pengguna dapat menggunakan aplikasi. Serta, guna dari foto yang ditampilkan untuk menunjukkan bahwa Expense Tracker adalah sebuah aplikasi yang dapat digunakan untuk membantu pengguna dalam mencatat, memantau, dan menganalisis pengeluaran seseorang atau suatu kelompok dalam rentang waktu tertentu.

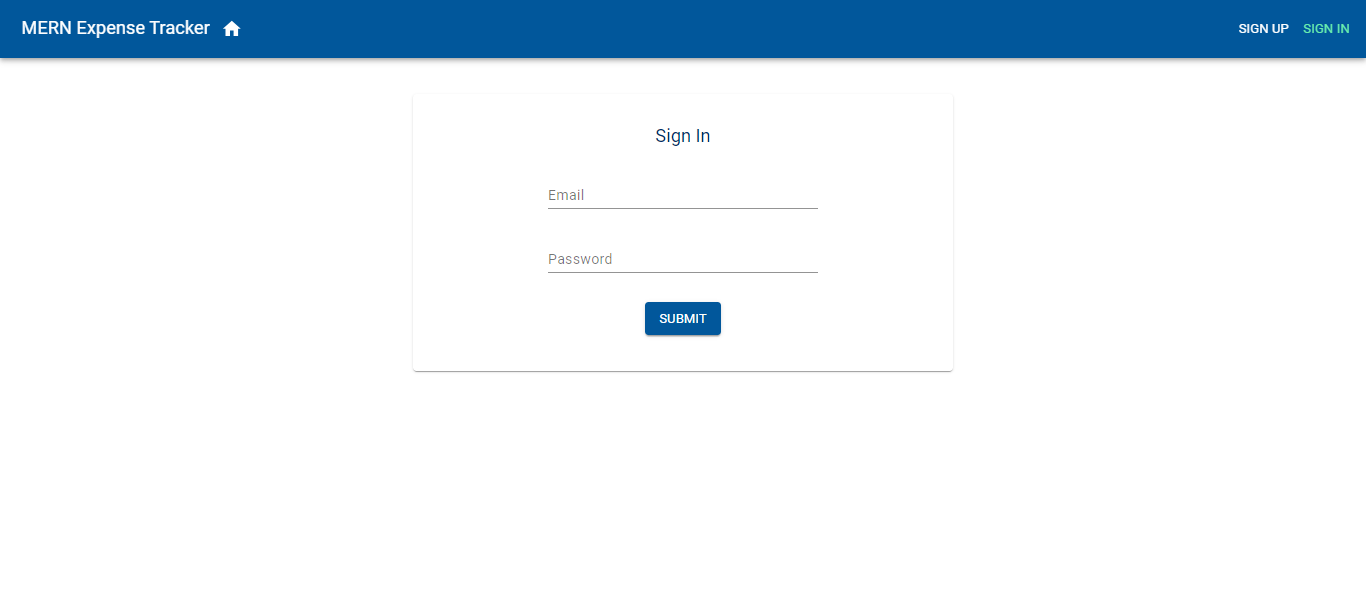
#### B. Halaman Sign Up



Gambar 2.37 Halaman Sign Up

Pada halaman Sign in memiliki beberapa label seperti label nama, email, dan password. Label-label tersebut dapat digunakan untuk melaukakan pendaftaran sebuah akun agar aplikasi dapat digunakan oleh pengguna. Fungsi dari halaman Sign up (daftar) adalah untuk memungkinkan pengguna baru untuk membuat akun pada aplikasi Expense Tracker ini, dan kiranya pengguna dapat memberikan informasi pribadi seperti nama, alamat email, dan kata sandi. Setelah pengguna sudah mendaftar maka pengguna bisa langsung menggunakan aplikasi dan mengakses fitur-fitur yang ada dalam aplikasi.

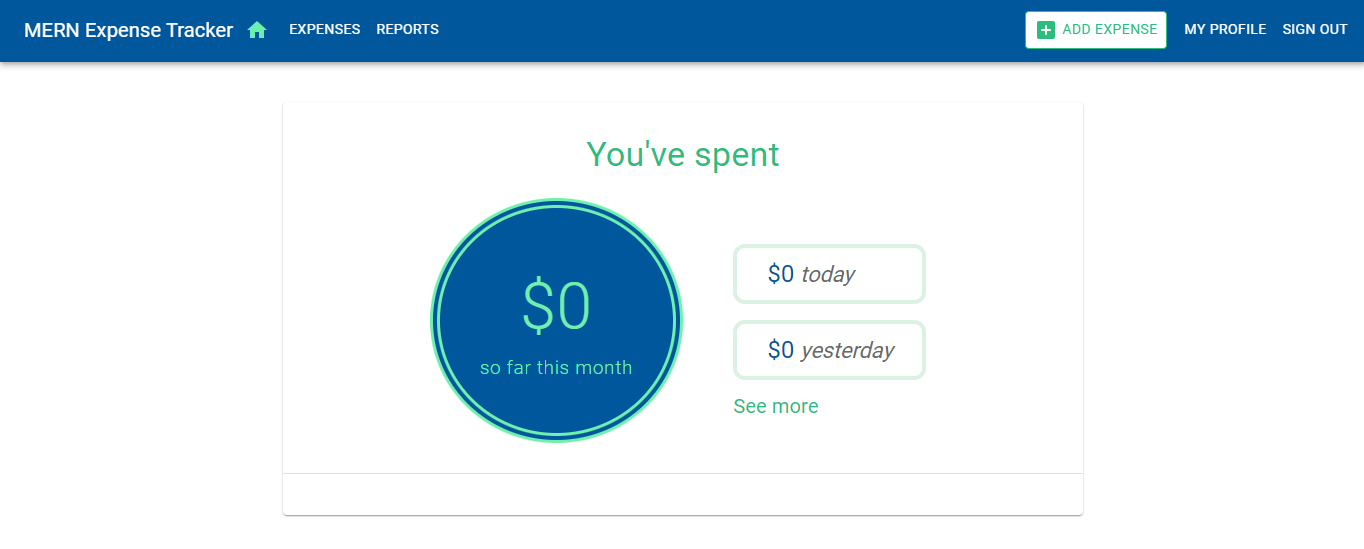
#### C. Halaman Sign In



Gambar 2.38 Halaman Sign In

Fungsi dari halaman Sign In (Masuk) pada aplikasi ini adalah untuk memudahkan pengguna yang sebelumnya sudah mendaftar atau pengguna lama untuk langsung masuk kedalam aplikasi. Pengguna hanya diminta untuk memasukan alamat email dan kata sandi untuk mengotentikasi identitas pengguna. Setelah berhasil masuk pengguna dapat mengakses semua fitur yang ada.

#### D. Halaman Beranda



Gambar 2.39 Halaman Sesudah Sign In

Pada halaman ini, Expense Tracker ini memiliki tata letak yang sederhana dan mudah dipahami. Elemen-elemennya tersusun rapi dalam sebuah kotak dengan latar belakang putih. Di bagian atas halaman terdapat logo aplikasi dan tombol navigasi. Di bagian tengah halaman terdapat informasi tentang pengeluaran pengguna, dan di bagian bawah halaman terdapat tombol untuk menambahkan pengeluaran baru.

Halaman Pengeluaran MERN Expense Tracker memiliki beberapa fungsi utama, yaitu:

* Menampilkan informasi tentang pengeluaran pengguna, termasuk pengeluaran hari ini, pengeluaran bulan ini, dan pengeluaran kemarin.
* Memungkinkan pengguna untuk menambahkan pengeluaran baru.
* Memungkinkan pengguna untuk melihat laporan pengeluaran.



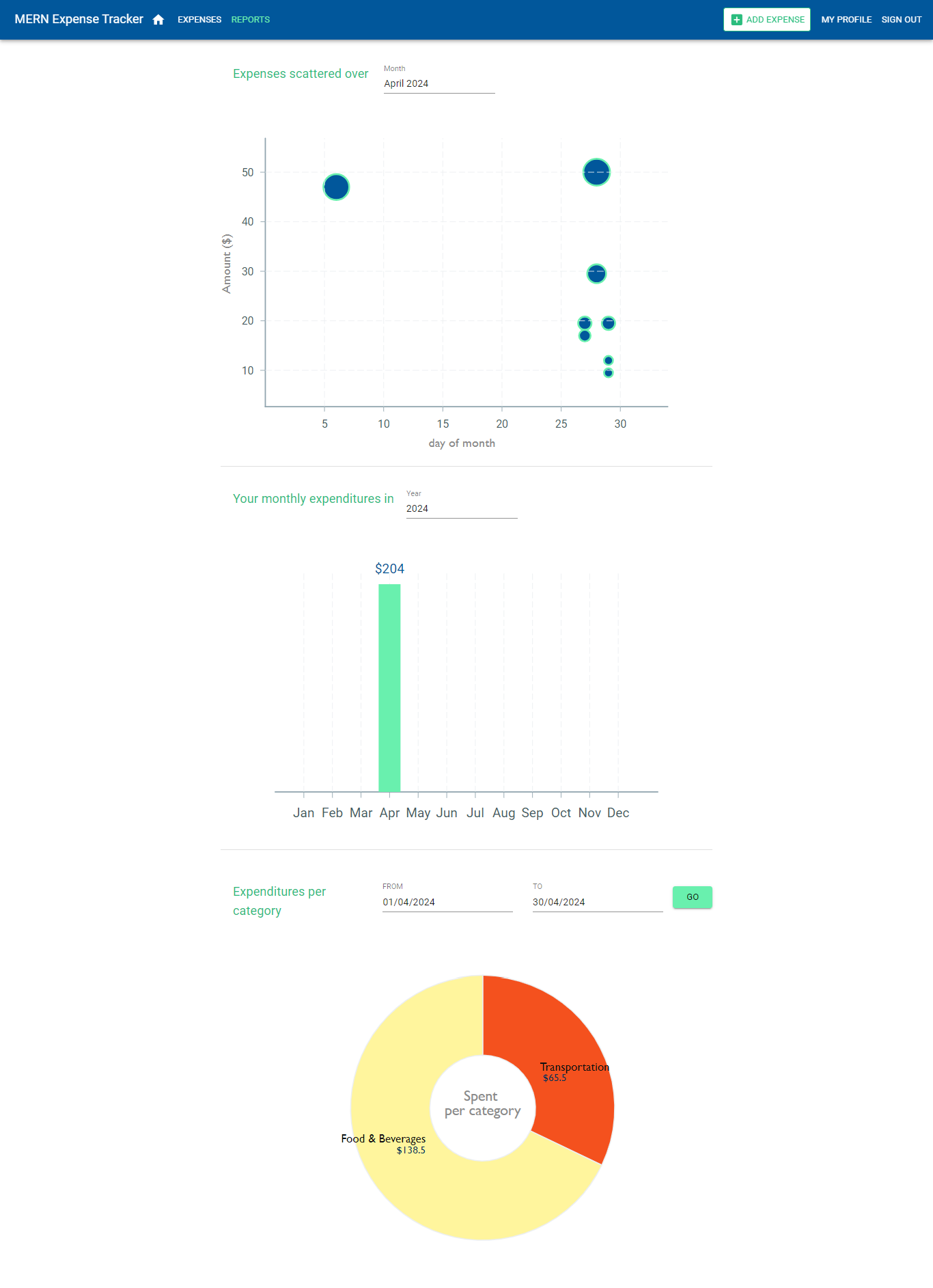
Gambar 2.40 Expenses Button

Expenses di dalam aplikasi pelacak berfungsi untuk membantu pengguna dalam memantau dan mengelola pengeluaran mereka. Fitur ini memungkinkan pengguna untuk mencatat dan melacak semua biaya yang mereka keluarkan, mulai dari belanja harian hingga tagihan bulanan.



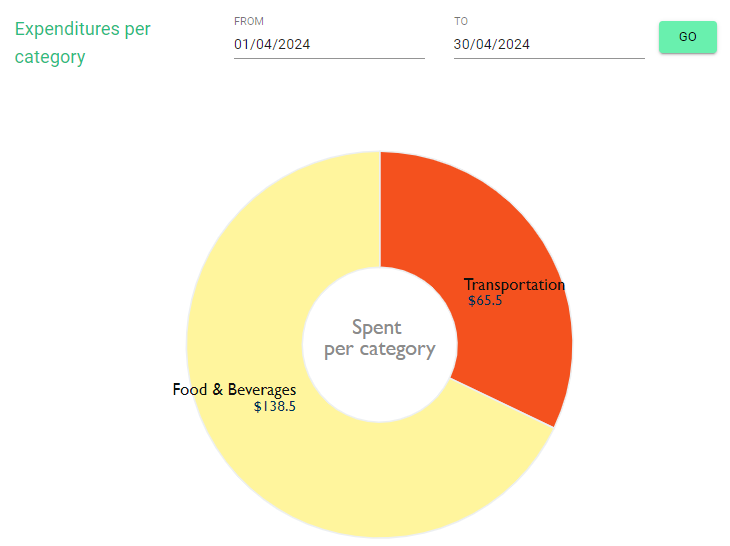
Gambar 2.41 Reports Button

Reports di dalam aplikasi pelacak berfungsi untuk menyajikan informasi yang terperinci dan terstruktur tentang aktivitas atau progres pengguna dalam mencapai tujuan mereka. Fitur ini memungkinkan pengguna untuk melihat ringkasan atau analisis dari data yang telah mereka rekam



Gambar 2.42 Halaman Beranda

Bagian ini menampilkan keseluruhan beranda yang terdapat juga beberapa diagram seperti diagram tebar atau *scatter diagram* yang dimana diagram ini berfungsi agar pengguna dapat menganalisi seberapa banyak pengeluaran yang terjadi dalam beberapa hari, dalam diagram tersebut juga memberitahukan pada setiap harinya pengguna telah mengeluarkan seberapa banyak uang untuk beberapa keperluan. Dan diagram selanjutnya adalah diagram bar, yang dimnana diagram ini menampilan jumlah total pengeluaran dalam satu bulan sang pengguna.Pada halaman paling bawah terdapat diagram pie yang dimana menampilakn beberapa kategori dan juga seberapa banya biaya yang telah dikeluarkan untuk kategori-kategori tersebut.

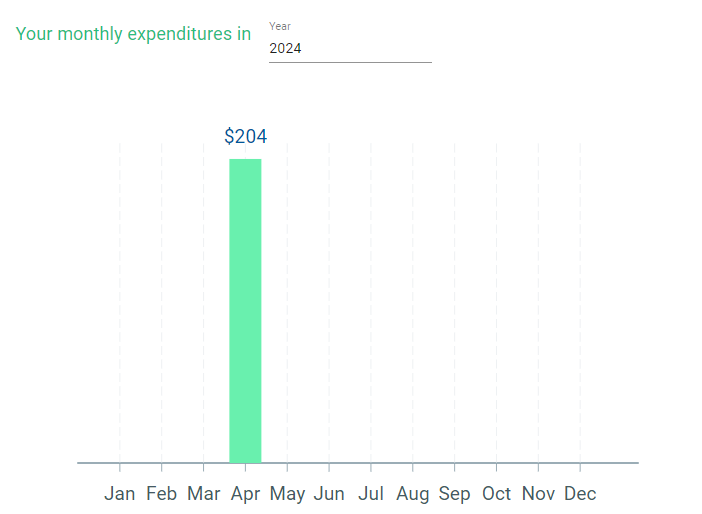


Gambar 2.43 Diagram Pie

Bagian ini menampilakan sebuah grafik lingkatan atau juga *pie chard* yang dimana menunjukkan seberapa banyak penggeluaran yang telah dihabiskan dalam bentuk kategori yang telah dibuat oleh pengguna untuk beberapa kebutuhan agar pengguna dapat mengetahui bahwa pengeluaran tersebut digunakan untuk keperluan apa saja. Halaman Laporan Pengeluaran mudah digunakan dan dipahami. Pengguna dapat dengan mudah melihat tabel pengeluaran mereka dan memfilter, mengurutkan, dan mengekspor data pengeluaran mereka. Halaman ini memungkinkan pengguna untuk melacak pengeluaran mereka dengan lebih detail.

Halaman Laporan Pengeluaran memiliki beberapa fungsi utama, yaitu:

* Menampilkan tabel pengeluaran pengguna.
* Memungkinkan pengguna untuk memfilter pengeluaran berdasarkan kategori, tanggal, atau jumlah.
* Memungkinkan pengguna untuk mengurutkan pengeluaran berdasarkan kategori, tanggal, atau jumlah.
* Memungkinkan pengguna untuk mengekspor data pengeluaran ke format file CSV.

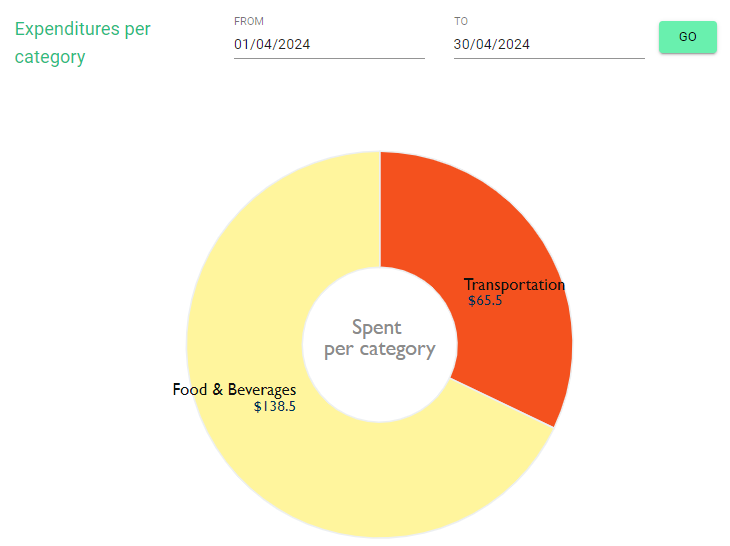


Gambar 2.44 Diagram Bar

Pada bagian ini menampilakan sebuah bar diagram yang dimana pengguna dapat mengetahuai sebarapa banyak pengeluaran yang terjadi dalam satu bulan penuh. Dan pengguna juga dapat dengan mudah melihat informasi tentang pengeluaran mereka dan menambahkan pengeluaran baru. Halaman ini juga memungkinkan pengguna untuk melihat laporan pengeluaran mereka, yang dapat membantu mereka mengelola keuangan mereka.

Halaman MERN Expense Tracker memiliki beberapa fungsi utama, yaitu:

* Menampilkan informasi tentang pengeluaran pengguna, termasuk pengeluaran hari ini, pengeluaran bulan ini, dan pengeluaran kemarin.
* Memungkinkan pengguna untuk menambahkan pengeluaran baru.
* Memungkinkan pengguna untuk melihat laporan pengeluaran.



Gambar 2.45 Category Expenses

Pada bagian ini pengguna dapat mengetahui seberapa banyak pengerluaran yang dilakukan dalam sebuah kategori, kategori tersebut dapat dipilih sesui dengan jangka waktu yang diinginkan pengguna. Dan pengguna dapat dengan mudah melihat tabel pengeluaran mereka dan memfilter, mengurutkan, dan mengekspor data pengeluaran mereka. Halaman ini memungkinkan pengguna untuk melacak pengeluaran mereka dengan lebih detail.

Halaman Laporan Pengeluaran memiliki beberapa fungsi utama, yaitu:

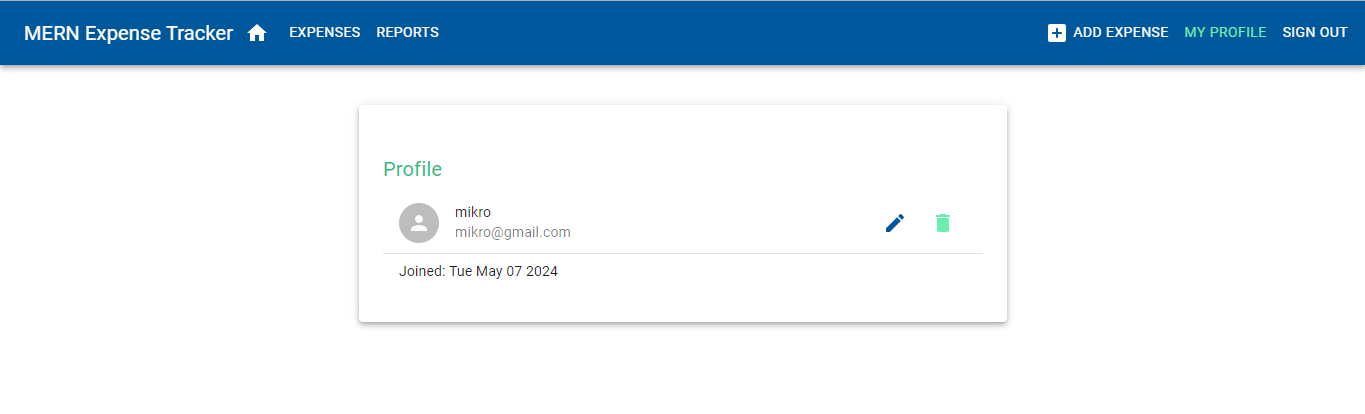
* Menampilkan tabel pengeluaran pengguna.
* Memungkinkan pengguna untuk memfilter pengeluaran berdasarkan kategori, tanggal, atau jumlah.
* Memungkinkan pengguna untuk mengurutkan pengeluaran berdasarkan kategori, tanggal, atau jumlah.
* Memungkinkan pengguna untuk mengekspor data pengeluaran ke format file CSV.



Gambar 2.46 Add Expense Button

Add expenses di dalam aplikasi pelacak berfungsi sebagai alat untuk mencatat dan memasukkan detail pengeluaran pengguna secara cepat dan efisien. Fitur ini memungkinkan pengguna untuk dengan mudah menambahkan informasi tentang biaya yang mereka keluarkan, seperti jumlah uang yang dihabiskan, deskripsi transaksi, dan kategori pengeluaran

#### E. Halaman My Profile



Gambar 2.47 Halaman My Profile

Fungsi dari halaman Sign In (Masuk) pada aplikasi ini adalah untuk memudahkan pengguna yang sebelumnya sudah mendaftar atau pengguna lama untuk langsung masuk kedalam aplikasi. Pengguna hanya diminta untuk memasukan alamat email dan kata sandi untuk mengotentikasi identitas pengguna. Setelah berhasil masuk pengguna dapat mengakses semua fitur yang ada. Tombol tanda pena menandakan bahwa tombol ini untuk mengedit, yang letaknya di posisi samping profile ini dapat memudahkan pengguna untuk mengerti fungsinya. Dan tulisan “Joined Tue May 07 2024”, memberikan informasi kapan pengguna membuat akun ini. Ikon tong sampah ini memberikan makna menghapus, dengan ini pengguna dapat langsung mengetahui fungsi dari ikon ini yang letaknya dekat dengan ikon pena dan profil.



Gambar 2.48 Edit Button

Edit di dalam aplikasi pelacak berfungsi sebagai alat yang memungkinkan pengguna untuk mengubah atau memperbarui informasi yang sudah ada. Dengan menekan tombol "edit", pengguna dapat membuka atau mengakses mode pengeditan di mana mereka dapat melakukan perubahan yang diperlukan pada data atau konten yang ada.



Gambar 2.49 Trash Button

Trash dalam aplikasi pelacak berfungsi sebagai alat untuk menghapus atau memindahkan item atau data yang tidak diinginkan atau tidak lagi diperlukan oleh pengguna. Fitur ini memberikan pengguna kemampuan untuk membersihkan atau mengelola ruang penyimpanan mereka dengan cara yang efisien dan efektif.

## 3.2.2.2 Coding

### A. Halaman Utama

Berikut ini *source code* untuk tampilan utama (home) sebelum memasukkan akun:



Source code yang membawa pengguna ke tampilan Home:

<Link to="/">

        <IconButton aria-label="Home" style={isActive(history, "/")}>

          <HomeIcon/>

        </IconButton>

      </Link>

1. Komponen Link

**<Link to="/">**

Link adalah komponen dari React Router yang digunakan untuk membuat navigasi berbasis URL dalam aplikasi React.

**to="/"**

berarti ketika link ini diklik, pengguna akan diarahkan ke halaman utama (root path) dari aplikasi.

2. Komponen IconButton

<IconButton aria-label="Home" style={isActive(history, "/")}>

IconButton adalah komponen dari Material-UI yang membuat tombol berbasis ikon.

**aria-label="Home"**

menambahkan label yang berguna untuk aksesibilitas, memberi tahu pembaca layar bahwa tombol ini adalah untuk "Home".

**style={isActive(history, "/")}**

menetapkan gaya pada tombol berdasarkan hasil fungsi isActive. Fungsi ini biasanya menentukan apakah rute saat ini aktif dan menambahkan gaya yang sesuai (misalnya, tombol di-highlight ketika pengguna berada di halaman tersebut).

3. Komponen HomeIcon

**<HomeIcon/>**

HomeIcon adalah ikon yang disediakan oleh Material-UI, yang biasanya berupa ikon rumah atau ikon lain yang relevan dengan beranda.

4. Penggunaan history dan isActive

history mengacu pada objek yang diberikan oleh React Router yang melacak riwayat navigasi.

**isActive(history, "/")**

adalah fungsi yang kemungkinan besar memeriksa apakah rute saat ini cocok dengan rute yang diberikan ("/" dalam hal ini) dan mengembalikan gaya yang sesuai (misalnya, mengubah warna atau latar belakang tombol).

Source code yang menjalankan tampilan Home yang akan dilihat pengguna sebelum signin:

export default function Home() {

  const classes = useStyles();

  return (

    <>

      {auth.isAuthenticated() && <ExpenseOverview />}

      {!auth.isAuthenticated() && typeof window !== "undefined" && (

        <Card className={classes.card}>

          <Typography variant="h6" className={classes.title}>

            Home Page

          </Typography>

          <CardMedia

            className={classes.media}

            image={expenseTrackerImg}

            title="Expense Tracker"

          />

          <Typography

            variant="body2"

            component="p"

            className={classes.credit}

            color="textSecondary"

          >

            Photo by{" "}

            <a

              href="https://unsplash.com/@anniespratt"

              target="\_blank"

              rel="noopener noreferrer"

            >

              Annie Spratt

            </a>{" "}

            on Unsplash

          </Typography>

          <CardContent>

            <Typography variant="body1" component="p">

              Welcome to the MERN Expense Tracker.{" "}

              <Link to="/signup">Sign up</Link> or{" "}

              <Link to="/signin">sign in</Link> to get started.

            </Typography>

          </CardContent>

        </Card>

      )}

    </>

  );

}

* + 1. Function *home*

export default function Home() {

Baris ini mengekspor fungsi “*Home*” sebagai ekspor default modul ini.

  const classes = useStyles();

Baris ini memanggil *hook* kustom (*hook* adalah fungsi yang dapat digunakan kembali) yang disebut `*useStyles*` untuk mendapatkan kelas CSS. Kelas-kelas ini digunakan untuk menata elemen-elemen komponen.

* + 1. Pernyataan *return*

  return (

Pernyataan untuk pengembalian fungsi, yang mengembalikan *JSX* untuk dirender.

      {auth.isAuthenticated() && <ExpenseOverview />}

Baris ini memeriksa apakah pengguna diautentikasi menggunakan metode `*auth.isAuthenticated()*`. Jika pengguna diautentikasi, ia akan merender dan menunjukkan komponen *<ExpenseOverview />.*

{!auth.isAuthenticated() && typeof window !== "undefined" && (

        <Card className={classes.card}>

Jika pengguna tidak diautentikasi dan kode berjalan di browser (bukan di server), kode akan dirender untuk menunjukkan beberapa informasi dan tautan bahwa pengguna tidak didefinisikan.

          <Typography variant="h6" className={classes.title}>

            Home Page

          </Typography>

Merender dan memanggil komponen *Typography* dengan varian "h6" (gaya header) dan kelas khusus untuk penataan gaya.

<CardMedia

            className={classes.media}

            image={expenseTrackerImg}

            title="Expense Tracker"

          />

Merender dan memanggil komponen *CardMedia*, yang digunakan untuk menampilkan konten media (misalnya gambar). Kelas `*classes.media*` digunakan untuk memanggil penataan gaya *media*, sumber gambar `e*xpenseTrackerIm*g`, dan *title*.

<Typography

            variant="body2"

            component="p"

            className={classes.credit}

            color="textSecondary"

          >

            Photo by{" "}

            <a

              href="https://unsplash.com/@anniespratt"

              target="\_blank"

              rel="noopener noreferrer"

            >

              Annie Spratt

            </a>{" "}

            on Unsplash

          </Typography>

Merender dan memanggil komponen *Typography* dengan varian “*body2*” dan kelas khusus. Komponenini berisi kredit untuk gambar tersebut, dengan link yang dapat diklik oleh pengguna untuk membawa mereka ke halaman Unsplash penulis.

<CardContent>

Merender dan memanggil komponen *CardContent*, yang biasanya berisi konten utama Kartu.

            <Typography variant="body1" component="p">

              Welcome to the MERN Expense Tracker.{" "}

              <Link to="/signup">Sign up</Link> or{" "}

              <Link to="/signin">sign in</Link> to get started.

            </Typography>

          </CardContent>

        </Card>

      )}

    </>

  );

}

Merender dan memanggil komponen *Typography* dengan varian *“body1”*. Komponen ini berisi pesan selamat datang serta tautan

(<Link to="/signup">Sign up</Link> or{" "} <Link to="/signin">sign in</Link> to get started.) yang akan digunakan oleh pengguna untuk membawa mereka kepada halaman *Sign up* dan *Sign in*

### B. Sign Up (Daftar)

Source code yang membawa user ke tampilan *Signup* ketika diklik:

 <Link to="/signup">

            <Button style={isActive(history, "/signup")}>Sign up

            </Button>

          </Link>

* + - * Link to

 <Link to="/signup">

Komponen Link dari React Router digunakan untuk membuat link yang dapat dinavigasi ke halaman “*/signup*”.

* + - * Sign Up button

Button style={isActive(history, "/signup")}>Sign up

            </Button>

          </Link>

Di dalam komponen *Link*, komponen *Button* akan menampilkan teks "*Sign Up*". Gaya atau *style* pada *Button* diatur menggunakan fungsi *isActive*, yang memeriksa apakah jalur saat ini cocok dengan */signup* dan menata tombol sesuai untuk menunjukkan bahwa jalur tersebut aktif.

Source code yang menjalankan tampilan atau *display signup* yang dilihat oleh pengguna:

export default function Signup() {

  const classes = useStyles()

  const [values, setValues] = useState({

    name: '',

    password: '',

    email: '',

    open: false,

    error: ''

  })

  const handleChange = name => event => {

    setValues({ ...values, [name]: event.target.value })

  }

  const clickSubmit = () => {

    const user = {

      name: values.name || undefined,

      email: values.email || undefined,

      password: values.password || undefined

    }

    create(user).then((data) => {

      if (data.error) {

        setValues({ ...values, error: data.error})

      } else {

        setValues({ ...values, error: '', open: true})

      }

    })

  }

    return (<div>

      <Card className={classes.card}>

        <CardContent>

          <Typography variant="h6" className={classes.title}>

            Sign Up

          </Typography>

          <TextField id="name" label="Name" className={classes.textField} value={values.name} onChange={handleChange('name')} margin="normal"/><br/>

          <TextField id="email" type="email" label="Email" className={classes.textField} value={values.email} onChange={handleChange('email')} margin="normal"/><br/>

          <TextField id="password" type="password" label="Password" className={classes.textField} value={values.password} onChange={handleChange('password')} margin="normal"/>

          <br/> {

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}</Typography>)

          }

        </CardContent>

* + Signup *function*

export default function Signup() {

Baris ini mendeklarasikan komponen *function* bernama *Signup* dan mengekspornya sebagai ekspor default dari modul, sehingga dapat diimpor dengan nama apa pun di file lain

  const classes = useStyles()

Baris ini menggunakan *hook custom* “*useStyles”* untuk memanggil nama kelas CSS yang akan menata elemen komponen.

  const [values, setValues] = useState({

    name: '',

    password: '',

    email: '',

    open: false,

    error: ''

  })

Baris ini mendeklarasikan nilai variabel status dan fungsi *setValues* ​​untuk meng-*update* nilai tersebut, menggunakan *useState*. Nilai diinisialisasi sebagai objek dengan properti untuk *name, password, email,* dan *error*.

  const handleChange = name => event => {

    setValues({ ...values, [name]: event.target.value })

  }

Baris ini mendefinisikan fungsi *handleChange* yang menggunakan *name* sebagai argumen dan mengembalikan fungsi lain. Fungsi yang dikembalikan mengambil peristiwa dan memperbarui properti terkait dalam nilai dengan nilai target peristiwa, sekaligus menjaga properti lainnya tidak berubah.

const clickSubmit = () => {

    const user = {

      name: values.name || undefined,

      email: values.email || undefined,

      password: values.password || undefined

    }

    create(user).then((data) => {

      if (data.error) {

        setValues({ ...values, error: data.error})

      } else {

        setValues({ ...values, error: '', open: true})

      }

    })

  }

Baris ini mendefinisikan fungsi *clickSubmit* yang membuat objek pengguna dengan *name, email* dan *passaword* dari *values*. Fungsi *Create*  (diasumsikan sebagai panggilan API untuk membuat pengguna baru) dipanggil dengan pengguna. Responsnya ditangani dengan *callback* berikut:

* Jika ada kesalahan, *values.error* disetel ke *data.error*.
* Jika tidak, *value.error* akan dihapus dan *value.open* disetel ke true.
  + *Return*

    return (<div>

      <Card className={classes.card}>

Kode *JSX* ini merender dan memanggil komponen *Signup*: komponen *Card* (dari Material-UI) dengan kelas dari *useStyles* untuk mengatur gaya komponen *Card* tersebut.

<CardContent>

          <Typography variant="h6" className={classes.title}>

            Sign Up

          </Typography>

          <TextField id="name" label="Name" className={classes.textField} value={values.name} onChange={handleChange('name')} margin="normal"/><br/>

          <TextField id="email" type="email" label="Email" className={classes.textField} value={values.email} onChange={handleChange('email')} margin="normal"/><br/>

          <TextField id="password" type="password" label="Password" className={classes.textField} value={values.password} onChange={handleChange('password')} margin="normal"/>

          <br/> {

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}</Typography>)

          }

        </CardContent>

CardContent berisi berbagai elemen formulir, yaitu:

          <Typography variant="h6" className={classes.title}>

            Sign Up

          </Typography>

* *Typography* yang menampilkan judul "Sign Up".

          <TextField id="name" label="Name" className={classes.textField} value={values.name} onChange={handleChange('name')} margin="normal"/><br/>

          <TextField id="email" type="email" label="Email" className={classes.textField} value={values.email} onChange={handleChange('email')} margin="normal"/><br/>

          <TextField id="password" type="password" label="Password" className={classes.textField} value={values.password} onChange={handleChange('password')} margin="normal"/>

          <br/> {

* Tiga komponen *TextField* untuk memasukkan *name, email* dan *password*, masing-masing dengan props yang sesuai untuk nilai, pengendali perubahan, dan gaya. <br/> yang berarti *linebreak* (memulai baris baru)

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}</Typography>)

          }

* Jika *values.error* benar (berarti ada kesalahan), komponen *Typography* menampilkan pesan kesalahan dengan *Icon* yang menunjukkan kesalahan.

Source code untuk button *Submit* pada form *Signup*, source code ini juga memberi kotak dialog kepada pengguna apabila akun telah sukses didaftarkan:

        <CardActions>

          <Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>

        </CardActions>

      </Card>

      <Dialog open={values.open} disableBackdropClick={true}>

        <DialogTitle>New Account</DialogTitle>

        <DialogContent>

          <DialogContentText>

            New account successfully created.

          </DialogContentText>

        </DialogContent>

1. *CardActions*

        <CardActions>

Komponen kontainer dari *Material-UI* yang akan digunakan untuk menempatkan tombol aksi untuk komponen *Card*. Komponen ini akan digunakan untuk menambahkan tombol seperti "*Submit*", "*Cancel*", dan lain-lain.

          <Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>

        </CardActions>

Bagian ini membuat tombol di dalam wadah *CardActions*. Ketika pengguna mengklik tombol "*Submit*", fungsi *clickSubmit* akan dipicu.

1. *Dialog*

      <Dialog open={values.open} disableBackdropClick={true}>

        <DialogTitle>New Account</DialogTitle>

        <DialogContent>

          <DialogContentText>

            New account successfully created.

          </DialogContentText>

        </DialogContent>

Bagian ini membuat modal *dialog*. *Dialog* ini dikendalikan oleh *open property* dari *values.open*. Jika *value.open* benar, *dialog* akan ditampilkan. *Dialog* tersebut memiliki judul "*New Account*" dan area konten dengan teks "*New account successfully created*."

Source code ini membawa pengguna ke tampilan *Signin* agar pengguna bisa masuk ke akun baru mereka:

        <DialogActions>

          <Link to="/signin">

            <Button color="primary" autoFocus="autoFocus" variant="contained">

              Sign In

            </Button>

          </Link>

        </DialogActions>

      </Dialog>

    </div>

    )

}

1. *DialogActions*

        <DialogActions>

Wadah *DialogActions* ini akan menampung tombol tindakan yang terkait dengan *dialog*.

          <Link to="/signin">

Membuat *hyperlink* ke rute “/*signin”* aplikasi. Saat pengguna mengeklik tautan ini, aplikasi akan menavigasi ke halaman *signin* tanpa memuat ulang seluruh aplikasi.

            <Button color="primary" autoFocus="autoFocus" variant="contained">

              Sign In

            </Button>

*Color, autoFocus* dan *variant* akan *style* tombol dan berisi teks "*Sign In*". Ketika diklik, melakukan navigasi karena berada di dalam komponen *Link*.

### C. Sign In (Masuk)

Source code *signin button* yang membawa pengguna ke tampilan *Sign In*:

<Link to="/signin">

            <Button style={isActive(history, "/signin")}>Sign In

            </Button>

          </Link>

1. Link to

 <Link to="/signup">

Komponen Link dari React Router digunakan untuk membuat link yang dapat dinavigasi ke halaman “*/signup*”.

1. Sign Up button

Button style={isActive(history, "/signup")}>Sign up

            </Button>

          </Link>

Di dalam komponen *Link*, komponen *Button* akan menampilkan teks "*Sign Up*". Gaya atau *style* pada *Button* diatur menggunakan fungsi *isActive*, yang memeriksa apakah jalur saat ini cocok dengan */signup* dan menata tombol sesuai untuk menunjukkan bahwa jalur tersebut aktif.

Metode fetch *signin* ini akan membuat request *POST* untuk verifikasi akun pengguna dengan backend, source code ini terletak pada file *api-auth*.js :

const signin = async (user) => {

  try {

    let response = await fetch('/auth/signin/', {

      method: 'POST',

      headers: {

        'Accept': 'application/json',

        'Content-Type': 'application/json'

      },

      credentials: 'include',

      body: JSON.stringify(user)

    })

    return await response.json()

  } catch(err) {

    console.log(err)

  }

}

export {

  signin,

signout

}

1. *Fetch signin*

const signin = async (user) => {

Baris ini mendefinisikan fungsi asinkron bernama signin yang mengambil parameter pengguna.

  try {

Blok *try* digunakan untuk menangani kesalahan apa pun yang mungkin terjadi selama eksekusi kode di dalamnya.

    let response = await fetch('/auth/signin/', {

      method: 'POST',

      headers: {

        'Accept': 'application/json',

        'Content-Type': 'application/json'

      },

      credentials: 'include',

      body: JSON.stringify(user)

    })

Permintaan *HTTP POST* dibuat kepada *endpoint*  */auth/signin/* menggunakan *API fetch.* Kata kunci *await* digunakan untuk menunggu sebuah *Promise* yang akan dikembalikan melalui *fetch* yang diselesaikan*.*:

* *method: 'POST'*: Menentukan bahwa ini adalah permintaan POST.
* *headers*: Berisi header untuk permintaan:
* *Accept: application/json'*: Menunjukkan bahwa klien mengharapkan respons JSON.
* *'Content-Type': 'application/json'*: Menunjukkan bahwa isi permintaan dalam format JSON.
* *credentials: 'include'*: Menunjukkan bahwa *cookie* harus disertakan dengan permintaan, *cookies* sering kali diperlukan untuk otentikasi.
* *body: JSON.stringify(user):* Mengubah objek pengguna menjadi string JSON untuk dikirim sebagai isi permintaan.

    return await response.json()

Setelah permintaan pengambilan selesai, respons diurai sebagai JSON menggunakan *response.json(),* yang juga mengembalikan sebuah *Promise*. Kata kunci *await* menunggu *Promise* ini diselesaikan, lalu objek JavaScript yang dihasilkan dikembalikan dari fungsi *signin*.

  } catch(err) {

    console.log(err)

  }

}

Jika ada kesalahan yang terjadi selama eksekusi blok *try*, kesalahan tersebut ditangkap di blok *catch* dan dicatat ke konsol.

export {

  signin,

signout

}

Baris ini mengekspor fungsi *signin* dan fungsi *signout* (dari folder lain). Hal ini memungkinkan fungsi-fungsi ini untuk diimpor dan digunakan di bagian lain aplikasi.

Source code yang menjalan form *Signin*:

export default function Signin(props) {

  const classes = useStyles();

  const [values, setValues] = useState({

    email: "",

    password: "",

    error: "",

    redirectToReferrer: false,

  });

  const clickSubmit = () => {

    const user = {

      email: values.email || undefined,

      password: values.password || undefined,

    };

    signin(user).then((data) => {

      if (data.error) {

        setValues({ ...values, error: data.error });

      } else {

        auth.authenticate(data, () => {

          setValues({ ...values, error: "", redirectToReferrer: true });

        });

      }

    });

  };

  const handleChange = (name) => (event) => {

    setValues({ ...values, [name]: event.target.value });

  };

  const { from } = props.location.state || {

    from: {

      pathname: "/",

    },

  };

  const { redirectToReferrer } = values;

  if (redirectToReferrer) {

    return <Redirect to={from} />;

  }

  return (

    <Card className={classes.card}>

      <CardContent>

        <Typography variant="h6" className={classes.title}>

          Sign In

        </Typography>

        <TextField

          id="email"

          type="email"

          label="Email"

          className={classes.textField}

          value={values.email}

          onChange={handleChange("email")}

          margin="normal"

        />

        <br />

        <TextField

          id="password"

          type="password"

          label="Password"

          className={classes.textField}

          value={values.password}

          onChange={handleChange("password")}

          margin="normal"

        />

        <br />{" "}

        {values.error && (

          <Typography component="p" color="error">

            <Icon color="error" className={classes.error}>

              error

            </Icon>

            {values.error}

          </Typography>

        )}

      </CardContent>

1. *Sign In function*

export default function Signin(props) {

Baris ini mendefinisikan dan mengekspor komponen React fungsional bernama Signin.

  const classes = useStyles();

  const [values, setValues] = useState({

    email: "",

    password: "",

    error: "",

    redirectToReferrer: false,

  });

Menyiapkan nilai variabel status dengan nilai awal untuk *email*, *password, error*, dan *redirectToReferrer*. *setValues* adalah fungsi yang akan memperbarui status ini.

1. *clickSubmit*

  const clickSubmit = () => {

    const user = {

      email: values.email || undefined,

      password: values.password || undefined,

    };

    signin(user).then((data) => {

      if (data.error) {

        setValues({ ...values, error: data.error });

      } else {

        auth.authenticate(data, () => {

          setValues({ ...values, error: "", redirectToReferrer: true });

        });

      }

    });

  };

Fungsi ini dipanggil ketika pengguna mengirimkan formulir. Fungsi ini menciptakan objek pengguna dengan *email* dan *password* dari *state*. Dan kemudian memanggil fungsi *signin* (yang sudah ditentukan sebelumnya) dengan objek pengguna.

* Jika proses *signin* menghasilkan *error*, status diperbarui untuk menyertakan *error message*.
* Jika proses *signin* berhasil, *auth.authenticate* dipanggil dan status diperbarui untuk menunjukkan diperlukannya pengalihan.

1. *handleChange*

  const handleChange = (name) => (event) => {

    setValues({ ...values, [name]: event.target.value });

  };

Fungsi ini mengembalikan fungsi lain yang memperbarui status nilai untuk bidang tertentu (*email* atau *password*) saat pengguna mengetik di bidang input.

1. *Redirection*

  const { from } = props.location.state || {

    from: {

      pathname: "/",

    },

  };

  const { redirectToReferrer } = values;

  if (redirectToReferrer) {

    return <Redirect to={from} />;

  }

Kode ini memeriksa apakah pengguna harus dialihkan setelah *signin* berhasil. Ini mengekstrak lokasi dari *props.location.state* (atau default ke jalur *root*). Jika *redirectToReferrer* benar, maka pengguna akan dialihkan ke lokasi asal menggunakan komponen *Redirect* dari *react-router-dom*.

1. Form *Signin*

  return (

    <Card className={classes.card}>

      <CardContent>

        <Typography variant="h6" className={classes.title}>

          Sign In

        </Typography>

        <TextField

          id="email"

          type="email"

          label="Email"

          className={classes.textField}

          value={values.email}

          onChange={handleChange("email")}

          margin="normal"

        />

        <br />

        <TextField

          id="password"

          type="password"

          label="Password"

          className={classes.textField}

          value={values.password}

          onChange={handleChange("password")}

          margin="normal"

        />

        <br />{" "}

        {values.error && (

          <Typography component="p" color="error">

            <Icon color="error" className={classes.error}>

              error

            </Icon>

            {values.error}

          </Typography>

        )}

      </CardContent>

Bagian kode ini mengembalikan struktur JSX komponen dan membuat kartu dengan:

        <Typography variant="h6" className={classes.title}>

          Sign In

        </Typography>

* Judulnya "Masuk".

        <TextField

          id="email"

          type="email"

          label="Email"

          className={classes.textField}

          value={values.email}

          onChange={handleChange("email")}

          margin="normal"

        />

        <br />

        <TextField

          id="password"

          type="password"

          label="Password"

          className={classes.textField}

          value={values.password}

          onChange={handleChange("password")}

          margin="normal"

        />

        <br />{" "}

* Dua kolom teks untuk *email* dan *password*, yang menggunakan fungsi *handleChange* untuk memperbarui status input pengguna.

{values.error && (

          <Typography component="p" color="error">

            <Icon color="error" className={classes.error}>

              error

            </Icon>

            {values.error}

          </Typography>

        )}

      </CardContent>

* Jika ada kesalahan dalam status, pesan kesalahan akan ditampilkan di bawah *textfield-*nya.

Button *Submit*  yang pengguna gunakan untuk *Signin:*

      <CardActions>

        <Button

          color="primary"

          variant="contained"

          onClick={clickSubmit}

          className={classes.submit}

        >

          Submit

        </Button>

      </CardActions>

    </Card>

  );

}

1. *CardActions*

        <CardActions>

Komponen kontainer dari *Material-UI* yang akan digunakan untuk menempatkan tombol aksi untuk komponen *Card*. Komponen ini akan digunakan untuk menambahkan tombol seperti "*Submit*", "*Cancel*", dan lain-lain.

        <Button

          color="primary"

          variant="contained"

          onClick={clickSubmit}

          className={classes.submit}

        >

          Submit

        </Button>

      </CardActions>

    </Card>

  );

}

Bagian ini membuat tombol di dalam wadah *CardActions*. Ketika pengguna mengklik tombol "*Submit*", fungsi *clickSubmit* akan dipicu.

### D. Sign Out (Keluar)

Source code yang membiarkan pengguna keluar dari akun mereka:

<Button color="inherit" onClick={() => {

              auth.clearJWT(() => history.push('/'))

            }}>Sign out</Button>

        </span>)

Coding ini menampilkan button dengan teks "*Sign out*" dan menangani *Click Event "onClick".* Ketika button diketik, *auth.clearJWT* dipanggil untuk menghapus *JWT* (logout pengguna). “*history.push('/')*” digunakan untuk mengarahkan pengguna ke beranda*.*

Metode *fetch signout* digunakan untuk membuat *request GET* untuk memanggil *API* *signout, source code* ini terletak pada api-auth.js:

const signout = async () => {

  try {

    let response = await fetch('/auth/signout/', { method: 'GET' })

    return await response.json()

  } catch(err) {

    console.log(err)

  }

}

export {

  signin,

  signout

}

* 1. *Fetch signout*

const signout = async () => {

Baris ini mendefinisikan fungsi yang disebut signout yang tidak memerlukan argumen apa pun. Kata kunci *async* memungkinkan penggunaan “*await*” dalam fungsi ini, yang digunakan untuk menangani operasi *asynchronous* (hal-hal yang memerlukan waktu, seperti mengambil data dari server).

  try {

Mulai dengan blok *try*, yang berisi kode yang akan menangkap kesalahan.

    let response = await fetch('/auth/signout/', { method: 'GET' })

Baris ini mengirimkan permintaan *GET* ke server di *endpoint* */auth/signout/* untuk mengeluarkan pengguna. “*await*” memastikan bahwa kode menunggu di sini hingga server merespons.

    return await response.json()

Setelah server merespons, baris ini mengubah respons ke format *JSON* dan mengembalikannya. Sekali lagi, “*await*” membuat kode menunggu hingga respons sepenuhnya dikonversi.

  } catch(err) {

Jika ada kesalahan yang terjadi pada blok try, blok *catch* akan menangkap kesalahan tersebut.

    console.log(err)

  }

}

Baris ini mencetak kesalahan ke konsol, sehingga dapat *error* yang terjadi.

* 1. *Export*

export {

  signin,

  signout

}

Baris ini mengekspor fungsi *signin* dan *signout* sehingga dapat digunakan di file lain, seperti file *signout* ini.

### E. Sign Out (Keluar)

Source code yang membawa pengguna ke tampilan Expenses:

{

          auth.isAuthenticated() && (<span>

            <Link to={"/expenses/all"}>

              <Button style={isActive(history, "/expenses/all")}>Expenses</Button>

            </Link>

          </span>)

        }

* 1. *Authenticate*

{

          auth.isAuthenticated() && (<span>

Baris ini menggunakan kurung kurawal {} untuk menyematkan *JavaScript* di dalam *JSxX* dan memeriksa apakah pengguna diautentikasi dengan memanggil fungsi *auth.isAuthenticated()*. Operator *&&* memastikan bahwa kode *JSX* hanya dirender jika *auth.isAuthenticated()* dikembalikan dengan nilai *true*.

* 1. *Link*

            <Link to={"/expenses/all"}>

Baris ini menggunakan komponen *Link* dari *react-router-dom* untuk membuat tautan yang navigasi ke URL *"/expenses/all*". Ketika diklik, pengguna akan dibawa ke halaman */expenses/all* tanpa memuat ulang seluruh halaman.

* 1. *Button*

              <Button style={isActive(history, "/expenses/all")}>Expenses</Button>

            </Link>

          </span>)

        }

Baris ini merender komponen *Button* dari *Material-UI* dengan label *Expenses.* Atribut style menerapkan gaya secara dinamis dengan memanggil fungsi *isActive*, yang memeriksa apakah rute saat ini cocok dengan *"/expenses/all"*. Jika cocok, maka akan dikembalikan gaya halaman yang menunjukkan bahwa *button* tersebut aktif.

Source code yang menjalankan tampilan *expenses* yang pengguna lihat:

export default function Expenses() {

    const classes = useStyles()

    const [redirectToSignin, setRedirectToSignin] = useState(false)

    const [saved, setSaved] = useState(false)

    const [error, setError] = useState('')

    const [expenses, setExpenses] = useState([])

    const jwt = auth.isAuthenticated()

    const date = new Date(), y = date.getFullYear(), m = date.getMonth()

    const [firstDay, setFirstDay] = useState(new Date(y, m, 1))

    const [lastDay, setLastDay] = useState(new Date(y, m + 1, 0))

    useEffect(() => {

        const abortController = new AbortController()

        const signal = abortController.signal

        listByUser({firstDay: firstDay, lastDay: lastDay},{t: jwt.token}, signal).then((data) => {

          if (data.error) {

            setRedirectToSignin(true)

          } else {

            setExpenses(data)

          }

        })

        return function cleanup(){

          abortController.abort()

        }

    }, [])

    const handleSearchFieldChange = name => date => {

        if(name=='firstDay'){

            setFirstDay(date)

        }else{

            setLastDay(date)

        }

    }

    const searchClicked = () => {

        listByUser({firstDay: firstDay, lastDay: lastDay},{t: jwt.token}).then((data) => {

            if (data.error) {

              setRedirectToSignin(true)

            } else {

              setExpenses(data)

            }

        })

    }

    const handleChange = (name, index) => event => {

        const updatedExpenses = [...expenses]

        updatedExpenses[index][name] = event.target.value

        setExpenses(updatedExpenses)

    }

    const handleDateChange = index => date => {

        const updatedExpenses = [...expenses]

        updatedExpenses[index].incurred\_on = date

        setExpenses(updatedExpenses)

      }

    const clickUpdate = (index) => {

        let expense = expenses[index]

        update({

            expenseId: expense.\_id

          }, {

            t: jwt.token

          }, expense).then((data) => {

            if (data.error) {

              setError(data.error)

            } else {

              setSaved(true)

              setTimeout(()=>{setSaved(false)}, 3000)

            }

        })

    }

    const removeExpense = (expense) => {

        const updatedExpenses = [...expenses]

        const index = updatedExpenses.indexOf(expense)

        updatedExpenses.splice(index, 1)

        setExpenses(updatedExpenses)

    }

    if (redirectToSignin) {

        return <Redirect to='/signin'/>

    }

    return (

      <div className={classes.root}>

      <div className={classes.search}>

      <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DatePicker

                    disableFuture

                    format="dd/MM/yyyy"

                    label="SHOWING RECORDS FROM"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={firstDay}

                    onChange={handleSearchFieldChange('firstDay')}

                />

                <DatePicker

                    format="dd/MM/yyyy"

                    label="TO"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={lastDay}

                    onChange={handleSearchFieldChange('lastDay')}

                />

        </MuiPickersUtilsProvider>

        <Button variant="contained" color="secondary" onClick={searchClicked}>GO</Button>

        </div>

      {expenses.map((expense, index) => {

            return   <span key={index}>

        <ExpansionPanel className={classes.panel}>

          <ExpansionPanelSummary

            expandIcon={<Edit />}

          >

            <div className={classes.info}>

                <Typography className={classes.amount}>$ {expense.amount}</Typography><Divider style={{marginTop: 4, marginBottom: 4}}/>

                <Typography>

                    {expense.category}

                </Typography>

                <Typography className={classes.date}>{new Date(expense.incurred\_on).toLocaleDateString()}</Typography>

            </div>

            <div>

                <Typography className={classes.heading}>{expense.title}</Typography>

                <Typography className={classes.notes}>

                    {expense.notes}

                </Typography>

            </div>

          </ExpansionPanelSummary>

          <Divider/>

          <ExpansionPanelDetails style={{display: 'block'}}>

          <div>

              <TextField label="Title" className={classes.textField} value={expense.title} onChange={handleChange('title', index)} margin="normal"/>

             <TextField label="Amount ($)" className={classes.textField} value={expense.amount} onChange={handleChange('amount', index)} margin="normal" type="number"/>

          </div>

          <div>

          <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DateTimePicker

                    label="Incurred on"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={expense.incurred\_on}

                    onChange={handleDateChange(index)}

                    showTodayButton

                />

          </MuiPickersUtilsProvider>

          <TextField label="Category" className={classes.textField} value={expense.category} onChange={handleChange('category', index)} margin="normal"/>

          </div>

          <TextField

            label="Notes"

            multiline

            rows="2"

            value={expense.notes}

            onChange={handleChange('notes', index)}

            className={classes.textField}

            margin="normal"

          />

          <div className={classes.buttons}>

          {

            error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {error}</Typography>)

          }

          {

              saved && <Typography component="span" color="secondary" className={classes.status}>Saved</Typography>

          }

            <Button color="primary" variant="contained" onClick={()=> clickUpdate(index)} className={classes.submit}>Update</Button>

            <DeleteExpense expense={expense} onRemove={removeExpense}/>

          </div>

          </ExpansionPanelDetails>

        </ExpansionPanel>

        </span>

        })}

      </div>

    )

  }

1. *Expenses function*

export default function Expenses() {

Baris ini mendefinisikan komponen fungsional *React* baru yang disebut *Expenses*. Ekspor default artinya komponen ini adalah ekspor default file, sehingga dapat diimpor ke file lain.

    const classes = useStyles()

Baris ini memanggil hook khusus *useStyles* untuk mendapatkan kelas *CSS* untuk menata gaya komponen. *classes* adalah objek yang berisi gaya.

    const [redirectToSignin, setRedirectToSignin] = useState(false)

    const [saved, setSaved] = useState(false)

    const [error, setError] = useState('')

    const [expenses, setExpenses] = useState([])

Baris berikut menggunakan kait useState untuk membuat beberapa variabel *state* yaitu:

* *redirectToSignin*: Menentukan apakah akan mengarahkan pengguna ke halaman masuk.
* *saved*: Melacak apakah pengeluaran telah berhasil disimpan.
* *error*: Menyimpan pesan kesalahan apa pun.
* *expenses*: Array untuk menyimpan daftar pengeluaran.

    const jwt = auth.isAuthenticated()

    const date = new Date(), y = date.getFullYear(), m = date.getMonth()

    const [firstDay, setFirstDay] = useState(new Date(y, m, 1))

    const [lastDay, setLastDay] = useState(new Date(y, m + 1, 0))

* *jwt*: Mengambil *JSON Web Token (JWT)* untuk memeriksa apakah pengguna diautentikasi.
* *date, y, m*: Membuat objek *Date* baru dan ekstrak tahun dan bulan saat ini.
* *firstDay*: Variabel status diinisialisasi ke hari pertama bulan ini.
* *lastDay*: Variabel status diinisialisasi ke hari terakhir bulan ini.

1. *useEffect*

    useEffect(() => {

        const abortController = new AbortController()

        const signal = abortController.signal

        listByUser({firstDay: firstDay, lastDay: lastDay},{t: jwt.token}, signal).then((data) => {

          if (data.error) {

            setRedirectToSignin(true)

          } else {

            setExpenses(data)

          }

        })

        return function cleanup(){

          abortController.abort()

        }

    }, [])

Efek ini berjalan ketika komponen dipasang:

* *abortController* dan *signal*: Digunakan untuk membatalkan permintaan pengambilan jika komponen dilepas.
* *listByUser*: Mengambil pengeluaran antara hari pertama dan hari terakhir. Jika ada kesalahan, maka *redirectToSignin* akan disetel ke true. Jika tidak, status pengeluaran akan diperbarui dengan data yang diambil.
* *cleanup*: Membatalkan permintaan pengambilan jika komponen dilepas.

1. *handSearchFieldChange*

    const handleSearchFieldChange = name => date => {

        if(name=='firstDay'){

            setFirstDay(date)

        }else{

            setLastDay(date)

        }

    }

Memperbarui variabel status tanggal:

* *name*: Menentukan tanggal yang akan diperbarui (*firstDay* atau *lastDay*).
* *date*: Tanggal baru yang akan ditetapkan.

1. *searchClicked*

    const searchClicked = () => {

        listByUser({firstDay: firstDay, lastDay: lastDay},{t: jwt.token}).then((data) => {

            if (data.error) {

              setRedirectToSignin(true)

            } else {

              setExpenses(data)

            }

        })

    }

Mengambil *expense* berdasarkan tanggal yang diperbarui ketika *search button* diklik. *listByUser* memanggil API untuk mengambil *expenses* dan memperbarui *state*-nya.

1. *handleChange*

    const handleChange = (name, index) => event => {

        const updatedExpenses = [...expenses]

        updatedExpenses[index][name] = event.target.value

        setExpenses(updatedExpenses)

    }

Memperbarui rincian *expense* di *state* ketika pengguna mengubah kolom input:

* *name*: Bidang yang sedang diperbarui (misalnya, judul, jumlah).
* *index*: Indeks biaya yang diperbarui.
* *event*: Peristiwa yang dipicu oleh perubahan input.

1. *handleDateChange*

    const handleDateChange = index => date => {

        const updatedExpenses = [...expenses]

        updatedExpenses[index].incurred\_on = date

        setExpenses(updatedExpenses)

      }

Memperbarui tanggal terjadinya pengeluaran:

* *index*: Indeks biaya yang diperbarui.
* *date*: Nilai tanggal baru

1. *clickUpdate*

    const clickUpdate = (index) => {

        let expense = expenses[index]

        update({

            expenseId: expense.\_id

          }, {

            t: jwt.token

          }, expense).then((data) => {

            if (data.error) {

              setError(data.error)

            } else {

              setSaved(true)

              setTimeout(()=>{setSaved(false)}, 3000)

            }

        })

    }

Mengirimkan data pengeluaran terkini ke server:

* *index*: Indeks biaya yang diperbarui.
* *expense*: Memanggil API untuk memperbarui pengeluaran dan menangani responsnya.

1. *removeExpense*

    const removeExpense = (expense) => {

        const updatedExpenses = [...expenses]

        const index = updatedExpenses.indexOf(expense)

        updatedExpenses.splice(index, 1)

        setExpenses(updatedExpenses)

    }

Menghapus *expense* dari *state*:

* *expense*: Objek pengeluaran yang akan dihapus.
* *updateExpenses.splice(index, 1)*: Menghapus suatu *expense* pada indeks tertentu.

1. *RedirectToSignin*

    if (redirectToSignin) {

        return <Redirect to='/signin'/>

    }

Variabel *redirectToSignin* mengindikasikan apakah pengguna perlu diarahkan ke halaman masuk (*signin*). *<Redirect to='/signin'/>* menyatakan bahwa jika *redirectToSignin* bernilai true, pengguna akan diarahkan ke halaman '/*signin'*.

1. *Root div*

    return (

      <div className={classes.root}>

*<div className={classes.root}>* membungkus seluruh komponen dalam sebuah *div* dengan kelas CSS *root* yang ditentukan dalam objek *classes*.

1. *Searchbar*

      <div className={classes.search}>

*<div className={classes.search}>* membungkus bagian pencarian dalam sebuah *div* dengan kelas CSS *search*.

      <MuiPickersUtilsProvider utils={DateFnsUtils}>

*<MuiPickersUtilsProvider utils={DateFnsUtils}>* menyediakan utilitas tanggal (*DateFns*) untuk komponen picker tanggal.

                <DatePicker

                    disableFuture

                    format="dd/MM/yyyy"

                    label="SHOWING RECORDS FROM"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={firstDay}

                    onChange={handleSearchFieldChange('firstDay')}

                />

                <DatePicker

                    format="dd/MM/yyyy"

                    label="TO"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={lastDay}

                    onChange={handleSearchFieldChange('lastDay')}

                />

        </MuiPickersUtilsProvider>

* *<DatePicker> <DatePicker/>* adalah komponen yang digunakan untuk memilih rentang tanggal.
* *disableFuture*: Menghalangi pemilihan tanggal di masa depan.
* *format*="dd/MM/yyyy": Menentukan format tampilan tanggal.
* *value={firstDay} dan value={lastDay}*: Mengatur nilai awal dan akhir dari tanggal yang dipilih.
* *onChange={handleSearchFieldChange('firstDay')}* dan *onChange={handleSearchFieldChange('lastDay')}* menangani perubahan pada input tanggal

        <Button variant="contained" color="secondary" onClick={searchClicked}>GO</Button>

        </div>

*Button* untuk memicu pencarian pengeluaran dalam rentang tanggal yang dipilih (*searchClicked*).

1. *Expenses*

      {expenses.map((expense, index) => {

            return   <span key={index}>

*expenses.map* digunakan untuk menghasilkan elemen untuk setiap pengeluaran dalam daftar expenses. Setiap elemen pengeluaran dikemas dalam *span* dengan kunci unik (*key*).

        <ExpansionPanel className={classes.panel}>

*ExpansionPanel* dari *Material-UI* digunakan untuk setiap *expense* hingga memungkinkan pengguna untuk memperluas atau menyembunyikan detail suatu *expense*.

          <ExpansionPanelSummary

            expandIcon={<Edit />}

          >

            <div className={classes.info}>

                <Typography className={classes.amount}>$ {expense.amount}</Typography><Divider style={{marginTop: 4, marginBottom: 4}}/>

                <Typography>

                    {expense.category}

                </Typography>

                <Typography className={classes.date}>{new Date(expense.incurred\_on).toLocaleDateString()}</Typography>

            </div>

            <div>

                <Typography className={classes.heading}>{expense.title}</Typography>

                <Typography className={classes.notes}>

                    {expense.notes}

                </Typography>

            </div>

          </ExpansionPanelSummary>

*ExpansionPanelSummary*: Menampilkan ringkasan pengeluaran dengan:

* *div* pertama (*classes.info*) yang berisi jumlah, kategori, dan tanggal pengeluaran.
* *div* kedua berisi judul *expense* dan *notes* *expense*.

          <Divider/>

          <ExpansionPanelDetails style={{display: 'block'}}>

*ExpansionPanelDetails* ditampilkan ketika panel diperluas dan berisi formulir untuk mengedit detail *expense.*

          <div>

              <TextField label="Title" className={classes.textField} value={expense.title} onChange={handleChange('title', index)} margin="normal"/>

             <TextField label="Amount ($)" className={classes.textField} value={expense.amount} onChange={handleChange('amount', index)} margin="normal" type="number"/>

          </div>

*<div>* pertama ini mengandung elemen input untuk judul dan jumlah *expense*, yaitu:

* *<TextField label="Title" className={classes.textField} value={expense.title} onChange={handleChange('title', index)} margin="normal"/>* adalah input untuk mengedit judul *expense*.
* *<TextField label="Amount ($)" className={classes.textField} value={expense.amount} onChange={handleChange('amount', index)} margin="normal" type="number"/>* adalah input untuk mengedit jumlah *expense*. Jenis input ini adalah input numerik.

          <div>

          <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DateTimePicker

                    label="Incurred on"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={expense.incurred\_on}

                    onChange={handleDateChange(index)}

                    showTodayButton

                />

          </MuiPickersUtilsProvider>

          <TextField label="Category" className={classes.textField} value={expense.category} onChange={handleChange('category', index)} margin="normal"/>

          </div>

*<div>* kedua ini mengandung elemen input untuk tanggal dan kategori *expense*.

* *<MuiPickersUtilsProvider utils={DateFnsUtils}>* adalah penyedia utilitas untuk bekerja dengan tanggal.
* *<DateTimePicker label="Incurred on" className={classes.textField} views={["year", "month", "date"]} value={expense.incurred\_on} onChange={handleDateChange(index)} showTodayButton />* adalah picker yang digunakan untuk memilih dan mengedit tanggal *expense*.
* *<TextField label="Category" className={classes.textField} value={expense.category} onChange={handleChange('category', index)} margin="normal"/>* adalah input yang digunakan untuk mengedit kategori *expense*.

          <TextField

            label="Notes"

            multiline

            rows="2"

            value={expense.notes}

            onChange={handleChange('notes', index)}

            className={classes.textField}

            margin="normal"

          />

Input yang digunakan untuk mengedit *note* *expense*, dengan *multiline* yang memungkinkan pengguna untuk menulis beberapa baris teks.

          <div className={classes.buttons}>

Bagian ini mengandung *button* dan pesan status.

          {

            error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {error}</Typography>)

          }

Coding ini akan menunjukkan pesan “*error”* apabila ada masalah dalam menyimpan hasil perubahan detail *expense.*

          {

              saved && <Typography component="span" color="secondary" className={classes.status}>Saved</Typography>

          }

Coding ini akan menunjukkan pesan “Saved” yang akan muncul apabila *expense* berhasil diedit atau diubah.

            <Button color="primary" variant="contained" onClick={()=> clickUpdate(index)} className={classes.submit}>Update</Button>

Coding untuk tombol “*Update*” yang akan memperbarui detail *expense*.

            <DeleteExpense expense={expense} onRemove={removeExpense}/>

          </div>

          </ExpansionPanelDetails>

        </ExpansionPanel>

        </span>

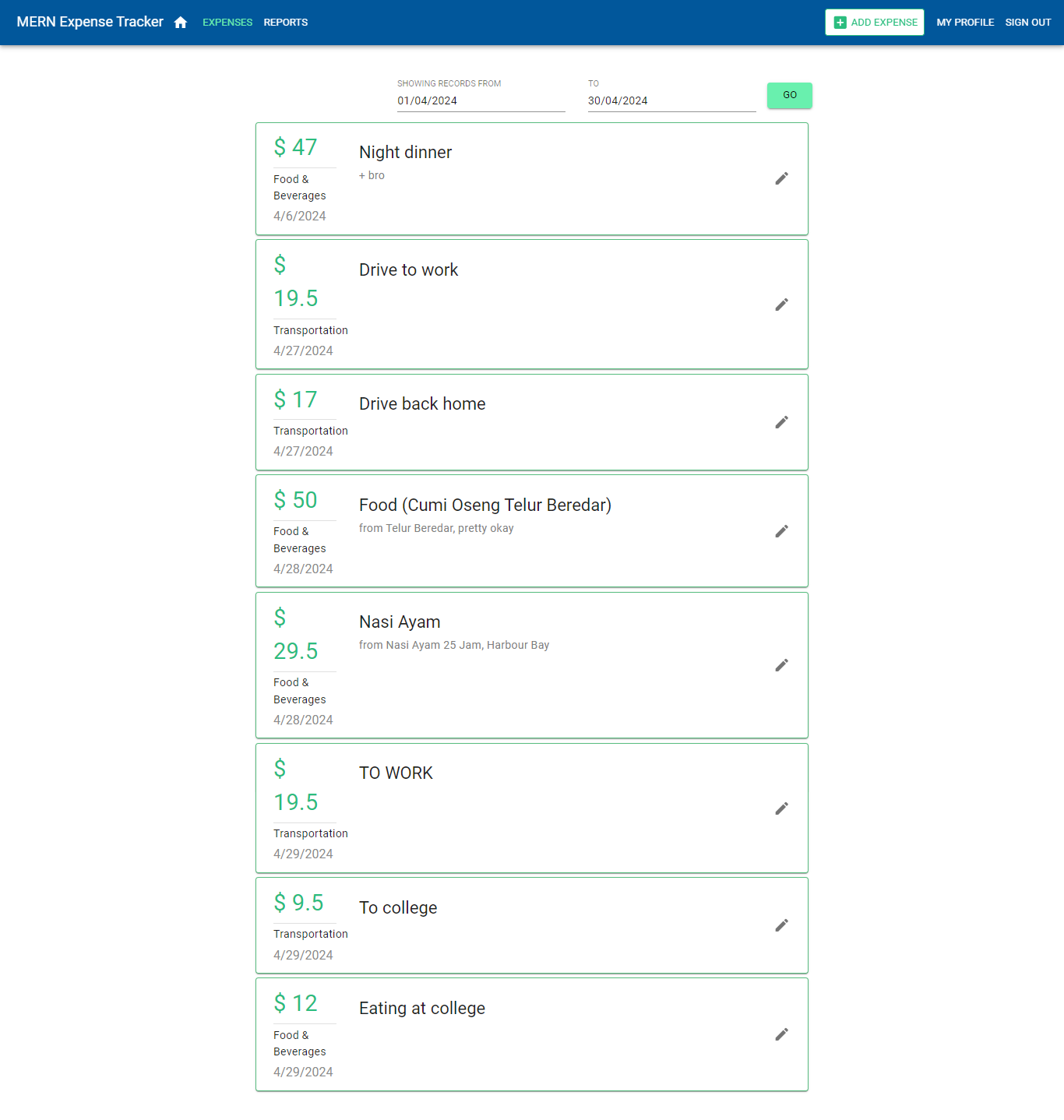
        })}

      </div>

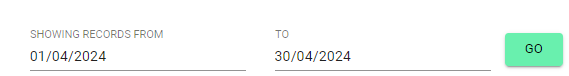
    )

  }

*DeleteExpense* adalah komponen yang akan digunakan untuk menghapus *expense* dengan memanggil *onRemove={removeExpense}* ketika pengguna menghapus *expense*.



Source code untuk filter expenses pada tanggal-tanggal tertentu:



    return (

      <div className={classes.root}>

      <div className={classes.search}>

      <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DatePicker

                    disableFuture

                    format="dd/MM/yyyy"

                    label="SHOWING RECORDS FROM"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={firstDay}

                    onChange={handleSearchFieldChange('firstDay')}

                />

                <DatePicker

                    format="dd/MM/yyyy"

                    label="TO"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={lastDay}

                    onChange={handleSearchFieldChange('lastDay')}

                />

        </MuiPickersUtilsProvider>

        <Button variant="contained" color="secondary" onClick={searchClicked}>GO</Button>

        </div>

1. *Filter expenses*

    return (

      <div className={classes.root}>

      <div className={classes.search}>

*<div className={classes.root}>* adalah elemen pembungkus utama yang menggunakan gaya dari kelas *classes.r oot*.

*<div className={classes.search}>* adalah wadah untuk komponen pencarian, menggunakan gaya dari kelas *classes.search*.

      <MuiPickersUtilsProvider utils={DateFnsUtils}>

*MuiPickersUtilsProvider*, penyedia utilitas untuk bekerja dengan komponen picker tanggal. *DateFnsUtils* adalah *library* yang digunakan untuk memanipulasi tanggal.

                <DatePicker

                    disableFuture

                    format="dd/MM/yyyy"

                    label="SHOWING RECORDS FROM"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={firstDay}

                    onChange={handleSearchFieldChange('firstDay')}

                />

                <DatePicker

                    format="dd/MM/yyyy"

                    label="TO"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={lastDay}

                    onChange={handleSearchFieldChange('lastDay')}

                />

        </MuiPickersUtilsProvider>

DatePicker: Komponen picker tanggal dari Material-UI.

* *disableFuture*: Prop ini mencegah pemilihan tanggal di masa depan.
* *format*: Format tanggal yang akan ditampilkan, dalam kasus ini *dd/MM/yyyy* (tanggal/bulan/tahun).
* *label*: *Label* yang ditampilkan di atas *picker*.
* *className*: Menggunakan gaya dari kelas *classes.textField*.
* *views*: Level tampilan yang tersedia dalam picker (*year, month, date*).
* *value*: Nilai saat ini dari picker (*firstDay* atau *lastDay*).
* *onChange*: Fungsi yang dipanggil saat nilai tanggal berubah (*handleSearchFieldChange*).

        <Button variant="contained" color="secondary" onClick={searchClicked}>GO</Button>

        </div>

Button adalah Tombol dari Material-UI yang memiliki *property:*

* *variant*: Menentukan gaya tombol, dalam kasus ini contained yang berarti tombol dengan latar belakang solid.
* *color*: Warna tombol, dalam kasus ini secondary.
* *onClick*: Fungsi yang dipanggil saat tombol diklik (searchClicked).

Source code untuk menampilkan list *expenses* pengguna:

      {expenses.map((expense, index) => {

            return   <span key={index}>

        <ExpansionPanel className={classes.panel}>

          <ExpansionPanelSummary

            expandIcon={<Edit />}

          >

            <div className={classes.info}>

                <Typography className={classes.amount}>$ {expense.amount}</Typography><Divider style={{marginTop: 4, marginBottom: 4}}/>

                <Typography>

                    {expense.category}

                </Typography>

                <Typography className={classes.date}>{new Date(expense.incurred\_on).toLocaleDateString()}</Typography>

            </div>

            <div>

                <Typography className={classes.heading}>{expense.title}</Typography>

                <Typography className={classes.notes}>

                    {expense.notes}

                </Typography>

            </div>

          </ExpansionPanelSummary>

          <Divider/>

          <ExpansionPanelDetails style={{display: 'block'}}>

          <div>

              <TextField label="Title" className={classes.textField} value={expense.title} onChange={handleChange('title', index)} margin="normal"/>

             <TextField label="Amount ($)" className={classes.textField} value={expense.amount} onChange={handleChange('amount', index)} margin="normal" type="number"/>

          </div>

          <div>

          <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DateTimePicker

                    label="Incurred on"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={expense.incurred\_on}

                    onChange={handleDateChange(index)}

                    showTodayButton

                />

          </MuiPickersUtilsProvider>

          <TextField label="Category" className={classes.textField} value={expense.category} onChange={handleChange('category', index)} margin="normal"/>

          </div>

          <TextField

            label="Notes"

            multiline

            rows="2"

            value={expense.notes}

            onChange={handleChange('notes', index)}

            className={classes.textField}

            margin="normal"

          />

          <div className={classes.buttons}>

          {

            error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {error}</Typography>)

          }

1. *Mapping expenses*

      {expenses.map((expense, index) => {

Code ini menggunakan fungsi *map* untuk mengiterasi (melakukan perulangan) melalui daftar *expenses*. Untuk setiap *expense* dalam daftar, akan dibuat elemen baru. Parameter *expense* mewakili data pengeluaran individu, dan *index* adalah indeks elemen dalam array.

1. *Span wrapper*

            return   <span key={index}>

Setiap elemen *expense* dibungkus dalam elemen *<span>.* Atribut *key={index}* diperlukan oleh *React* untuk melacak elemen-elemen daftar secara efisien.

1. *ExpansionPanel*

        <ExpansionPanel className={classes.panel}>

Komponen *ExpansionPanel* adalah wadah yang dapat diperluas yang menampilkan informasi pengeluaran. *className={classes.panel}* menambahkan gaya khusus dari objek *classes*.

1. *ExpansionPanelSummary*

          <ExpansionPanelSummary

            expandIcon={<Edit />}

          >

*ExpansionPanelSummary* adalah ringkasan dari panel yang dapat diperluas. *expandIcon={<Edit />}* menambahkan ikon edit untuk memperjelas bahwa elemen ini dapat diedit.

1. *Div Info*

            <div className={classes.info}>

*Div* yang berisi informasi dasar tentang *expense*, seperti jumlah, kategori, dan tanggal yang ditampilkan dengan gaya khusus.

                <Typography className={classes.amount}>$ {expense.amount}</Typography><Divider style={{marginTop: 4, marginBottom: 4}}/>

                <Typography>

                    {expense.category}

                </Typography>

                <Typography className={classes.date}>{new Date(expense.incurred\_on).toLocaleDateString()}</Typography>

            </div>

Komponen *Typography* digunakan untuk menampilkan teks dengan gaya tertentu. *Divider* digunakan untuk memisahkan elemen-elemen visual.

            <div>

                <Typography className={classes.heading}>{expense.title}</Typography>

                <Typography className={classes.notes}>

                    {expense.notes}

                </Typography>

            </div>

          </ExpansionPanelSummary>

          <Divider/>

Menampilkan judul dan catatan untuk setiap pengeluaran.

1. *ExpansionPanelDetails*

          <ExpansionPanelDetails style={{display: 'block'}}>

Bagian ini berisi detail yang lebih lengkap yang hanya terlihat saat panel diperluas. style={{display: 'block'}} mengatur gaya tampilan.

          <div>

              <TextField label="Title" className={classes.textField} value={expense.title} onChange={handleChange('title', index)} margin="normal"/>

             <TextField label="Amount ($)" className={classes.textField} value={expense.amount} onChange={handleChange('amount', index)} margin="normal" type="number"/>

          </div>

Komponen *TextField* digunakan untuk input teks yang dapat diedit. *label* menentukan *label* untuk *input*, *value* menampilkan nilai saat ini, dan *onChange* menangani perubahan nilai input.

          <div>

          <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DateTimePicker

                    label="Incurred on"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={expense.incurred\_on}

                    onChange={handleDateChange(index)}

                    showTodayButton

                />

          </MuiPickersUtilsProvider>

Komponen *DateTimePicker* digunakan untuk memilih tanggal dan waktu. *MuiPickersUtilsProvider* menyediakan utilitas yang dibutuhkan oleh *picker*.

          <TextField label="Category" className={classes.textField} value={expense.category} onChange={handleChange('category', index)} margin="normal"/>

          </div>

          <TextField

            label="Notes"

            multiline

            rows="2"

            value={expense.notes}

            onChange={handleChange('notes', index)}

            className={classes.textField}

            margin="normal"

          />

Komponen TextField untuk kategori dan catatan yang *multiline*.

<div className={classes.buttons}>

          {

            error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {error}</Typography>)

          }

Bagian yang menampilkan pesan *error* jika terjadi. Jika variabel error memiliki nilai, pesan kesalahan ditampilkan.

Source code untuk mengubah atau mengedit expense sudah dibuat:

          {

              saved && <Typography component="span" color="secondary" className={classes.status}>Saved</Typography>

          }

Coding ini menggunakan operator logika *&&* untuk melakukan rendering kondisional. Jika *saved* bernilai *true*, maka komponen *<Typography>* akan *dirender*, jika tidak, tidak ada yang dirender.

Komponen *Typography* dari *Material-UI* digunakan untuk menampilkan teks dengan gaya yang konsisten. *component="span"* menentukan bahwa *Typography* akan dirender sebagai elemen HTML *<span>. color="secondary"* menentukan warna teks sebagai warna sekunder yang telah didefinisikan dalam tema *Material-UI. className={classes.status}* menambahkan kelas CSS status yang didefinisikan dalam objek classes untuk memberikan gaya tambahan pada elemen.

Source code untuk button Update:

            <Button color="primary" variant="contained" onClick={()=> clickUpdate(index)} className={classes.submit}>Update</Button>

            <DeleteExpense expense={expense} onRemove={removeExpense}/>

          </div>

          </ExpansionPanelDetails>

        </ExpansionPanel>

        </span>

        })}

      </div>

    )

  }

1. *Update Button*

            <Button color="primary" variant="contained" onClick={()=> clickUpdate(index)} className={classes.submit}>Update</Button>

Komponen *Button* dari Material-UI digunakan untuk membuat tombol Update. Komponen *button* memiliki:

* color="primary": Menentukan warna utama untuk tombol.
* variant="contained": Menentukan gaya tombol agar tampak solid.
* onClick={() => clickUpdate(index)}: Menentukan fungsi yang akan dijalankan ketika tombol diklik. Dalam hal ini, fungsi clickUpdate dipanggil dengan index sebagai argumen.
* className={classes.submit}: Menambahkan kelas CSS submit untuk memberikan gaya khusus pada tombol.

1. *DeleteExpense*

<DeleteExpense expense={expense} onRemove={removeExpense}/>

Komponen *DeleteExpense* digunakan untuk menghapus pengeluaran. Kita asumsikan ini adalah komponen yang menerima dua props: expense dan onRemove:

* *expense={expense}* mengirimkan data pengeluaran saat ini ke komponen *DeleteExpense*.
* *onRemove={removeExpense}* menentukan fungsi yang akan dijalankan untuk menghapus *expense*.

Source code untuk menghapus expense tertentu:

    const removeExpense = (expense) => {

        const updatedExpenses = [...expenses]

        const index = updatedExpenses.indexOf(expense)

        updatedExpenses.splice(index, 1)

        setExpenses(updatedExpenses)

    }

    if (redirectToSignin) {

        return <Redirect to='/signin'/>

    }

<DeleteExpense expense={expense} onRemove={removeExpense}/>

1. *removeExpense*

    const removeExpense = (expense) => {

Mendeklarasikan fungsi bernama removeExpense. Fungsi ini menerima satu argumen bernama expense.

        const updatedExpenses = [...expenses]

Membuat salinan baru dari *array expenses* dengan menggunakan *spread operator (...)* untuk menyalin semua nilai dari *array expenses* ke array baru yang disebut *updatedExpenses*.

        const index = updatedExpenses.indexOf(expense)

Mencari indeks *expense* di dalam array *updatedExpenses* menggunakan metode *indexOf*. Metode ini mengembalikan indeks dari elemen pertama dalam *array* yang sama dengan nilai yang diberikan (*expense*). Jika tidak ada elemen yang cocok, ia mengembalikan -1.

        updatedExpenses.splice(index, 1)

Menghapus satu elemen dari array *updatedExpenses* pada indeks yang ditemukan sebelumnya menggunakan metode *splice*. Ini mengubah array itu sendiri.

        setExpenses(updatedExpenses)

    }

Menggunakan *setExpenses* untuk memperbarui *state expenses* dalam aplikasi dengan array *updatedExpenses* yang baru, sehingga memperbarui tampilan aplikasi dengan menghapus *expense* yang dipilih.

1. redirectToSignin

    if (redirectToSignin) {

        return <Redirect to='/signin'/>

    }

Coding penggunaan kondisional. Jika *redirectToSignin* benar (*true*), maka komponen akan dialihkan (*redirect*) ke halaman *sign-in*. Ini digunakan untuk mengarahkan pengguna ke halaman *sign-in* jika mereka belum masuk.

1. *DeleteExpense*

<DeleteExpense expense={expense} onRemove={removeExpense}/>

Komponen bernama *DeleteExpense* yang menerima properti *expense* dan *onRemove*. *expense* adalah pengeluaran yang akan dihapus, dan *onRemove* adalah fungsi yang akan dipanggil saat pengguna memilih untuk menghapus pengeluaran tersebut.

### F. Reports Menu

Source code untuk membawa pengguna ke tampilan Reports:

{

          auth.isAuthenticated() && (<span>

            <Link to={"/expenses/reports"}>

              <Button style={isActive(history, "/expenses/reports")}>Reports</Button>

            </Link>

          </span>)

        }

1. *Authenticate*

{

          auth.isAuthenticated() && (<span>

*auth.isAuthenticated()* adalah fungsi yang mengecek apakah pengguna sudah terotentikasi (*logged in*). Jika true, itu berarti pengguna sudah masuk/log in. Kode ini menggunakan *&&* (operator logika *AND*) untuk menentukan apakah bagian berikutnya dari kode harus dijalankan atau tidak. Jika *auth.isAuthenticated*() mengembalikan true, maka bagian dalam tanda kurung kurawal ({}) akan dieksekusi. *<span>* adalah elemen HTML yang digunakan sebagai pembungkus (wrapper) di sini.

1. *Link*

            <Link to={"/expenses/reports"}>

Link adalah komponen dari *React Router* yang membuat link atau tautan. Tautan ini akan mengarahkan pengguna ke URL "/expenses/reports".

1. *Button*

              <Button style={isActive(history, "/expenses/reports")}>Reports</Button>

            </Link>

          </span>)

        }

*<Button>* adalah komponen tombol. *style={isActive(history, "/expenses/reports")}* mengatur gaya (*style*) dari tombol tersebut, yang bergantung pada fungsi *isActive,* yang mengecek apakah rute saat ini cocok dengan *"/expenses/reports*".

Source code yang menjalankan tampilan Reports yang pengguna lihat:

export default function Reports() {

    const classes = useStyles()

    return (

      <div className={classes.root}>

        <MonthlyScatter />

        <Divider className={classes.separator}/>

        <YearlyBar/>

        <Divider className={classes.separator}/>

        <CategoryPie/>

        </div>

    )

}

1. *Reports Function*

export default function Reports() {

Fungsi komponen *React* bernama Reports. Komponen ini diekspor sebagai default, sehingga bisa diimpor dengan mudah di file lain dan mengembalikan elemen JSX yang menentukan apa yang akan ditampilkan di layar.

    const classes = useStyles()

*const classes = useStyles()* memanggil sebuah *hook* untuk mendapatkan gaya *classes* (CSS classes) yang digunakan untuk styling elemen-elemen di dalam komponen ini. *useStyles* adalah fungsi yang dibuat dengan bantuan pustaka seperti *Material-UI* untuk membuat gaya dinamis.

    return (

      <div className={classes.root}>

*Return* menentukan apa yang akan dirender/dipasang ke layar ketika komponen ini digunakan. *<div className={classes.root}>* membuat elemen *div* dengan kelas *root* dari classes, yang berfungsi sebagai pembungkus utama.

        <MonthlyScatter />

        <Divider className={classes.separator}/>

        <YearlyBar/>

        <Divider className={classes.separator}/>

        <CategoryPie/>

        </div>

    )

}

*<MonthlyScatter />, <YearlyBar />, dan <CategoryPie />* adalah komponen lain yang dimasukkan di dalam komponen *Reports*. Komponen ini adalah grafik atau bagan yang berbeda yang akan ditampilkan dalam laporan. *MonthlyScatter* mungkin adalah bagan atau grafik yang menunjukkan data bulanan. *YearlyBar* mungkin adalah diagram batang yang menunjukkan data tahunan. *CategoryPie* mungkin adalah diagram pie yang menunjukkan data berdasarkan kategori.

*<Divider className={classes.separator}/>* adalah komponen yang digunakan untuk membuat garis pemisah antara elemen-elemen di dalam *div*. *className={classes.separator}* memberikan gaya khusus untuk pemisah ini, kemungkinan untuk menambahkan jarak atau gaya tertentu.

### G. Expenses scattered over

Source code yang menampilkan *expenses* pengguna dalam bentuk diagram *scatter plot*:

export default function MonthlyScatter() {

    const classes = useStyles()

    const [error, setError] = useState('')

    const [plot, setPlot] = useState([])

    const [month, setMonth] = useState(new Date())

    const jwt = auth.isAuthenticated()

    useEffect(() => {

        const abortController = new AbortController()

        const signal = abortController.signal

        plotExpenses({month: month},{t: jwt.token}, signal).then((data) => {

          if (data.error) {

            setError(data.error)

          } else {

            setPlot(data)

          }

        })

        return function cleanup(){

          abortController.abort()

        }

    }, [])

    const handleDateChange = date => {

        setMonth(date)

        plotExpenses({month: date},{t: jwt.token}).then((data) => {

          if (data.error) {

            setError(data.error)

          } else {

            setPlot(data)

          }

        })

    }

1. *MonthlyScatter function*

export default function MonthlyScatter() {

    const classes = useStyles()

Komponen ini diekspor sebagai *default*, artinya bisa diimpor dengan nama apapun di file lain. Fungsi *useStyles* adalah hook untuk mengatur gaya (styling) komponen ini.

    const [error, setError] = useState('')

    const [plot, setPlot] = useState([])

    const [month, setMonth] = useState(new Date())

*Hook* *useState* digunakan untuk mendefinisikan tiga *state*:

* *error*: untuk menyimpan pesan kesalahan, jika ada.
* *plot*: untuk menyimpan data yang akan digunakan dalam *scatter plot*.
* *month*: untuk menyimpan bulan yang sedang dipilih, default-nya adalah bulan saat ini (tanggal sekarang).

1. *Authenticate*

    const jwt = auth.isAuthenticated()

Variabel *jwt* menyimpan token autentikasi yang diperlukan untuk melakukan permintaan data. *auth.isAuthenticated()* mengembalikan objek yang berisi token.

1. *useEffect*

    useEffect(() => {

        const abortController = new AbortController()

        const signal = abortController.signal

        plotExpenses({month: month},{t: jwt.token}, signal).then((data) => {

          if (data.error) {

            setError(data.error)

          } else {

            setPlot(data)

          }

        })

        return function cleanup(){

          abortController.abort()

        }

    }, [])

*Hook* *useEffect* digunakan untuk melakukan *side-effect*, seperti mengambil data dari server, saat komponen pertama kali dirender:

* Membuat *AbortController* untuk membatalkan permintaan jika komponen tidak lagi diperlukan (misalnya, ketika komponen di-unmount).
* *plotExpenses* adalah fungsi yang mengambil data pengeluaran berdasarkan bulan yang dipilih dan token autentikasi.
* Jika ada kesalahan dalam pengambilan data (*data.error*), pesan kesalahan disimpan tersebut di *error*.
* Jika tidak ada kesalahan, data yang diterima disimpan di plot.
* Fungsi *cleanup* meng-*abort* permintaan ketika komponen di-*unmount* atau sebelum *effect* dijalankan lagi.

1. *handDateChange*

    const handleDateChange = date => {

        setMonth(date)

        plotExpenses({month: date},{t: jwt.token}).then((data) => {

          if (data.error) {

            setError(data.error)

          } else {

            setPlot(data)

          }

        })

    }

Fungsi *handleDateChange* dipanggil saat pengguna mengubah bulan yang dipilih:

* Mengubah *state* month dengan bulan yang baru dipilih *(setMonth(date)).*
* Mengambil data pengeluaran untuk bulan baru tersebut dan memperbarui *state* *plot* atau *error* sesuai dengan hasilnya.

Source code untuk filter expenses di scatter plot berdasarkan bulan dan tahun:

    return (

      <div style={{marginBottom: 20}}>

      <Typography variant="h6" className={classes.title}>Expenses scattered over </Typography>

      <MuiPickersUtilsProvider utils={DateFnsUtils}>

                 <DatePicker value={month} onChange={handleDateChange} views={["year", "month"]}

                 disableFuture

                    label="Month"

                    animateYearScrolling

                    variant="inline"/>

          </MuiPickersUtilsProvider>

        <VictoryChart

                theme={VictoryTheme.material}

                height={400}

                width={550}

                domainPadding={40}

                >

                    <VictoryScatter

                        style={{

                            data: { fill: "#01579b", stroke: "#69f0ae", strokeWidth: 2 },

                            labels: { fill: "#01579b", fontSize: 10, padding:8}

                        }}

                        bubbleProperty="y"

                        maxBubbleSize={15}

                        minBubbleSize={5}

                        labels={({ datum }) => `$${datum.y} on ${datum.x}th`}

                        labelComponent={<VictoryTooltip/>}

                        data={plot}

                        domain={{x: [0, 31]}}

                    />

                    <VictoryLabel

                      textAnchor="middle"

                      style={{ fontSize: 14, fill: '#8b8b8b' }}

                      x={270} y={390}

                      text={`day of month`}

                    />

                    <VictoryLabel

                      textAnchor="middle"

                      style={{ fontSize: 14, fill: '#8b8b8b' }}

                      x={6} y={190}

                      angle = {270}

                      text={`Amount ($)`}

                    />

                </VictoryChart>

        </div>

    )

  }

1. *Filter Expense Scatter Plot*

    return (

      <div style={{marginBottom: 20}}>

Komponen ini mengembalikan sebuah elemen div dengan margin bawah sebesar 20 piksel.

      <Typography variant="h6" className={classes.title}>Expenses scattered over </Typography>

Komponen *Typography* dari *Material-UI* digunakan untuk menampilkan teks judul. *variant="h6"* menentukan ukuran dan gaya teks sesuai dengan tema *Material-UI*. *classes.title* berisi gaya CSS tambahan untuk teks ini.

      <MuiPickersUtilsProvider utils={DateFnsUtils}>

                 <DatePicker value={month} onChange={handleDateChange} views={["year", "month"]}

                 disableFuture

                    label="Month"

                    animateYearScrolling

                    variant="inline"/>

          </MuiPickersUtilsProvider>

*MuiPickersUtilsProvider* digunakan untuk menyediakan utilitas tanggal dari *DateFnsUtils*.

* *DatePicker* adalah komponen pemilih tanggal yang digunakan untuk memilih bulan. Properti yang diatur:
* *value={month}:* nilai tanggal saat ini yang dipilih.
* *onChange={handleDateChange}:* fungsi yang dipanggil saat tanggal diubah.
* *views={["year", "month"]}:* pengguna hanya bisa memilih tahun dan bulan.
* *disableFuture:* tidak memungkinkan memilih tanggal di masa depan.
* *label="Month":* label untuk pemilih tanggal.
* *animateYearScrolling*: mengaktifkan animasi saat menggulir tahun.
* *variant="inline":* pemilih tanggal ditampilkan inline.

        <VictoryChart

                theme={VictoryTheme.material}

                height={400}

                width={550}

                domainPadding={40}

                >

*VictoryChart* adalah komponen dari victory untuk membuat chart.

* *theme={VictoryTheme.material}:* menggunakan tema material.
* *height={400}* dan *width={550}:* ukuran chart.
* *domainPadding={40}:* memberikan padding pada domain chart.

                    <VictoryScatter

                        style={{

                            data: { fill: "#01579b", stroke: "#69f0ae", strokeWidth: 2 },

                            labels: { fill: "#01579b", fontSize: 10, padding:8}

                        }}

                        bubbleProperty="y"

                        maxBubbleSize={15}

                        minBubbleSize={5}

                        labels={({ datum }) => `$${datum.y} on ${datum.x}th`}

                        labelComponent={<VictoryTooltip/>}

                        data={plot}

                        domain={{x: [0, 31]}}

                    />

*VictoryScatter* digunakan untuk membuat scatter plot.

* *style*: menentukan gaya untuk data dan label.
* *data*: warna dan gaya titik scatter.
* *labels*: warna dan gaya label.
* *bubbleProperty="y":* menentukan bahwa ukuran bubble berdasarkan nilai y.
* *maxBubbleSize* dan *minBubbleSize*: ukuran maksimum dan minimum *bubble*.
* *labels={({ datum }) => $${datum.y} on ${datum.x}th}:* format label yang ditampilkan saat hover.
* *labelComponent={<VictoryTooltip/>}:* menggunakan tooltip untuk label.
* *data={plot}:* data yang ditampilkan di scatter plot.
* *domain={{x: [0, 31]}}*: domain sumbu x dari 0 sampai 31 (hari dalam sebulan).

                    <VictoryLabel

                      textAnchor="middle"

                      style={{ fontSize: 14, fill: '#8b8b8b' }}

                      x={270} y={390}

                      text={`day of month`}

                    />

                    <VictoryLabel

                      textAnchor="middle"

                      style={{ fontSize: 14, fill: '#8b8b8b' }}

                      x={6} y={190}

                      angle = {270}

                      text={`Amount ($)`}

                    />

*VictoryLabel* digunakan untuk menampilkan teks label pada sumbu x dan y.

* textAnchor="middle": teks ditempatkan di tengah.
* style: menentukan gaya teks.
* *x* dan *y*: posisi label pada chart.
* *angle={270}:* memutar label sumbu y.
* *text*: teks yang ditampilkan pada label.

                </VictoryChart>

        </div>

    )

  }

Menutup elemen *VictoryChart* dan div yang membungkus keseluruhan *JSX*.

### H. Your monthly expenditures

Source code yang menampilkan *expenses* pengguna dalam bentuk diagram *bar plot*:

export default function Reports() {

    const classes = useStyles()

    const [error, setError] = useState('')

    const [year, setYear] = useState(new Date())

    const [yearlyExpense, setYearlyExpense] = useState([])

    const jwt = auth.isAuthenticated()

    const monthStrings = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

    useEffect(() => {

        const abortController = new AbortController()

        const signal = abortController.signal

        yearlyExpenses({year: year.getFullYear()},{t: jwt.token}, signal).then((data) => {

          if (data.error) {

            setError(data.error)

          }

            setYearlyExpense(data)

        })

        return function cleanup(){

          abortController.abort()

        }

    }, [])

    const handleDateChange = date => {

        setYear(date)

        yearlyExpenses({year: date.getFullYear()},{t: jwt.token}).then((data) => {

          if (data.error) {

            setError(data.error)

          }

            setYearlyExpense(data)

        })

    }

1. *Monthly Report Export*

export default function Reports() {

    const classes = useStyles()

Komponen *Reports* diekspor sebagai default, sehingga dapat diimpor dengan nama apapun di file lain. *useStyles* adalah *hook* yang mengatur gaya (styling) komponen ini.

    const [error, setError] = useState('')

    const [year, setYear] = useState(new Date())

    const [yearlyExpense, setYearlyExpense] = useState([])

Di sini hook *useState* digunakan untuk mendefinisikan tiga *state*:

* *error*: untuk menyimpan pesan kesalahan jika ada.
* *year*: untuk menyimpan tahun yang sedang dipilih, dengan nilai awal adalah tahun saat ini.
* *yearlyExpense*: untuk menyimpan data pengeluaran tahunan.

    const jwt = auth.isAuthenticated()

    const monthStrings = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

Variabel *jwt* menyimpan token autentikasi yang diperlukan untuk melakukan permintaan data. *auth*.*isAuthenticated*() mengembalikan objek yang berisi token.

*monthStrings* adalah array yang berisi nama-nama bulan dalam bentuk singkatan, digunakan untuk menampilkan data dengan lebih mudah dimengerti.

1. *useEffect*

    useEffect(() => {

        const abortController = new AbortController()

        const signal = abortController.signal

        yearlyExpenses({year: year.getFullYear()},{t: jwt.token}, signal).then((data) => {

          if (data.error) {

            setError(data.error)

          }

            setYearlyExpense(data)

        })

        return function cleanup(){

          abortController.abort()

        }

    }, [])

*Hook useEffect* digunakan untuk melakukan side-effect, seperti mengambil data dari server, saat komponen pertama kali dirender:

* Membuat *AbortController* untuk membatalkan permintaan jika komponen tidak lagi diperlukan.
* *yearlyExpenses* adalah fungsi yang mengambil data pengeluaran berdasarkan tahun yang dipilih dan token autentikasi.
* Jika ada kesalahan dalam pengambilan data (*data.error*), pesan kesalahan tersebut disimpan di *error*.
* Jika tidak ada kesalahan, data yang diterima disimpan di *yearlyExpense*.
* Fungsi *cleanup* meng-*abort* permintaan ketika komponen di-*unmount* atau sebelum effect dijalankan lagi.

1. *handleDateChange*

    const handleDateChange = date => {

        setYear(date)

        yearlyExpenses({year: date.getFullYear()},{t: jwt.token}).then((data) => {

          if (data.error) {

            setError(data.error)

          }

            setYearlyExpense(data)

        })

    }

Fungsi *handleDateChange* dipanggil saat pengguna mengubah tahun yang dipilih:

* Mengubah *state* *year* dengan tahun yang baru dipilih *(setYear(date)).*
* Mengambil data pengeluaran untuk tahun baru tersebut dan memperbarui state *yearlyExpense* atau *error* sesuai dengan hasilnya.

Source code untuk filter expenses di scatter plot berdasarkan tahun:

    return (

      <div>

          <Typography variant="h6" className={classes.title}>Your monthly expenditures in</Typography>

          <MuiPickersUtilsProvider utils={DateFnsUtils}>

            <DatePicker value={year} onChange={handleDateChange} views={["year"]}

                disableFuture

                label="Year"

                animateYearScrolling

                variant="inline"/>

          </MuiPickersUtilsProvider>

          <VictoryChart

                theme={VictoryTheme.material}

                domainPadding={10}

                height={300}

                width={450}>

                <VictoryAxis/>

                <VictoryBar

                    categories={{

                        x: monthStrings

                    }}

                    style={{ data: { fill: "#69f0ae", width: 20 }, labels: {fill: "#01579b"} }}

                    data={yearlyExpense.monthTot}

                    x={monthStrings['x']}

                    domain={{x: [0, 13]}}

                    labels={({ datum }) => `$${datum.y}`}

                />

          </VictoryChart>

      </div>

    )

  }

1. *Filter Monthly Expenditures*

    return (

      <div>

Komponen ini mengembalikan sebuah elemen div yang berfungsi sebagai pembungkus untuk semua elemen lainnya.

          <Typography variant="h6" className={classes.title}>Your monthly expenditures in</Typography>

Komponen *Typography* dari *Material-UI* digunakan untuk menampilkan teks judul. *variant="h6"* menentukan ukuran dan gaya teks sesuai dengan tema *Material-UI. classes.title* mungkin berisi gaya *CSS* tambahan untuk teks ini.

          <MuiPickersUtilsProvider utils={DateFnsUtils}>

            <DatePicker value={year} onChange={handleDateChange} views={["year"]}

                disableFuture

                label="Year"

                animateYearScrolling

                variant="inline"/>

          </MuiPickersUtilsProvider>

*MuiPickersUtilsProvider* digunakan untuk menyediakan utilitas tanggal dari *DateFnsUtils*. *DatePicker* adalah komponen pemilih tanggal yang digunakan untuk memilih tahun. Properti yang diatur, berupa:

* *value={year}:* nilai tanggal saat ini yang dipilih.
* *onChange={handleDateChange}:* fungsi yang dipanggil saat tahun diubah.
* *views={["year"]}:* pengguna hanya bisa memilih tahun.
* *disableFuture*: tidak memungkinkan memilih tahun di masa depan.
* *label="Year":* label untuk pemilih tahun.
* *animateYearScrolling*: mengaktifkan animasi saat menggulir tahun.
* *variant="inline":* pemilih tahun ditampilkan inline.

          <VictoryChart

                theme={VictoryTheme.material}

                domainPadding={10}

                height={300}

                width={450}>

                <VictoryAxis/>

                <VictoryBar

                    categories={{

                        x: monthStrings

                    }}

                    style={{ data: { fill: "#69f0ae", width: 20 }, labels: {fill: "#01579b"} }}

                    data={yearlyExpense.monthTot}

                    x={monthStrings['x']}

                    domain={{x: [0, 13]}}

                    labels={({ datum }) => `$${datum.y}`}

                />

          </VictoryChart>

*VictoryChart* adalah komponen dari *victory* untuk membuat *chart*.

* *theme={VictoryTheme.material}:* menggunakan tema material.
* *domainPadding={10}:* memberikan padding pada domain chart.
* *height={300}* dan *width={450}:* ukuran chart.

*VictoryAxis*: Menambahkan sumbu x dan y pada chart. VictoryBar digunakan untuk membuat bar chart dengan *property*:

* *categories*: menentukan kategori untuk sumbu x dengan menggunakan array monthStrings.
* *style*: menentukan gaya untuk data dan label.
* *data*: warna dan lebar batang.
* *labels*: warna label.
* *data={yearlyExpense.monthTot}*: data yang ditampilkan di bar chart.
* *domain={{ x: [0, 13] }}:* domain sumbu x dari 0 sampai 13 (untuk menampung semua bulan).
* *labels={({ datum }) => $${datum.y}}*: format label yang ditampilkan pada setiap batang.

      </div>

    )

  }

Menutup elemen div yang membungkus keseluruhan JSX.

### I. Expenditures per category

Source code yang menampilkan expenses pengguna dalam bentuk diagaram *pie chart*:

export default function Reports() {

    const classes = useStyles()

    const [error, setError] = useState('')

    const [expenses, setExpenses] = useState([])

    const jwt = auth.isAuthenticated()

    const date = new Date(), y = date.getFullYear(), m = date.getMonth()

    const [firstDay, setFirstDay] = useState(new Date(y, m, 1))

    const [lastDay, setLastDay] = useState(new Date(y, m + 1, 0))

    useEffect(() => {

        const abortController = new AbortController()

        const signal = abortController.signal

        averageCategories({firstDay: firstDay, lastDay: lastDay},{t: jwt.token}, signal).then((data) => {

          if (data.error) {

            setError(data.error)

          } else {

            setExpenses(data)

          }

        })

        return function cleanup(){

          abortController.abort()

        }

    }, [])

    const handleDateChange = name => date => {

        if(name=='firstDay'){

            setFirstDay(date)

        }else{

            setLastDay(date)

        }

    }

    const searchClicked = () => {

        averageCategories({firstDay: firstDay, lastDay: lastDay},{t: jwt.token}).then((data) => {

            if (data.error) {

              setRedirectToSignin(true)

            } else {

              setExpenses(data)

            }

        })

    }

1. *Expenditures per Category Pie Chart Export Report*

export default function Reports() {

    const classes = useStyles()

Komponen ini diekspor sebagai *default*, sehingga dapat diimpor dengan nama apapun di file lain. *useStyles* mungkin adalah hook untuk mengatur gaya (*styling*) komponen ini.

    const [error, setError] = useState('')

    const [expenses, setExpenses] = useState([])

    const jwt = auth.isAuthenticated()

Di sini kita menggunakan hook useState untuk mendefinisikan tiga state:

* *error*: untuk menyimpan pesan kesalahan jika ada.
* *expenses*: untuk menyimpan data pengeluaran.
* *jwt*: menyimpan token autentikasi yang diperlukan untuk melakukan permintaan data. *auth*.*isAuthenticated*() mengembalikan objek yang berisi *token*.

    const date = new Date(), y = date.getFullYear(), m = date.getMonth()

    const [firstDay, setFirstDay] = useState(new Date(y, m, 1))

    const [lastDay, setLastDay] = useState(new Date(y, m + 1, 0))

* *date*: menyimpan tanggal saat ini.
* *y*: menyimpan tahun saat ini.
* *m*: menyimpan bulan saat ini.
* *firstDay*: state yang menyimpan hari pertama bulan ini.
* *lastDay*: state yang menyimpan hari terakhir bulan ini.

1. *useEffect*

    useEffect(() => {

        const abortController = new AbortController()

        const signal = abortController.signal

        averageCategories({firstDay: firstDay, lastDay: lastDay},{t: jwt.token}, signal).then((data) => {

          if (data.error) {

            setError(data.error)

          } else {

            setExpenses(data)

          }

        })

        return function cleanup(){

          abortController.abort()

        }

    }, [])

Hook *useEffect* digunakan untuk mengambil data dari server saat komponen pertama kali dirender:

* Membuat *AbortController* untuk membatalkan permintaan jika komponen tidak lagi diperlukan.
* *averageCategories* adalah fungsi yang mengambil data *expenses* berdasarkan tanggal awal (*firstDay*) dan tanggal akhir (*lastDay*) serta *token* autentikasi (*jwt*.*token*).
* Jika ada kesalahan dalam pengambilan data (*data.error*), pesan kesalahan tersebut disimpan di *error*.
* Jika tidak ada kesalahan, data yang diterima disimpan di *expenses*.
* Fungsi *cleanup* meng-*abort* permintaan ketika komponen di-unmount atau sebelum effect dijalankan lagi.

1. *handleDateChange*

    const handleDateChange = name => date => {

        if(name=='firstDay'){

            setFirstDay(date)

        }else{

            setLastDay(date)

        }

    }

Fungsi *handleDateChange* dipanggil saat pengguna mengubah tanggal:

* Jika name adalah *firstDay*, mengubah *state* *firstDay* dengan tanggal baru yang dipilih *(setFirstDay(date)).*
* Jika name adalah *lastDay*, mengubah *state* *lastDay* dengan tanggal baru yang dipilih *(setLastDay(date)).*

1. *searchClicked*

    const searchClicked = () => {

        averageCategories({firstDay: firstDay, lastDay: lastDay},{t: jwt.token}).then((data) => {

            if (data.error) {

              setRedirectToSignin(true)

            } else {

              setExpenses(data)

            }

        })

    }

Fungsi *searchClicked* dipanggil saat tombol pencarian diklik:

* Mengambil data *expenses* untuk rentang tanggal yang dipilih (*firstDay* dan *lastDay*) serta *token* autentikasi (*jwt.token*).
* Jika ada kesalahan dalam pengambilan data (*data.error*), pengguna akan diarahkan untuk *signin (setRedirectToSignin(true)).*
* Jika tidak ada kesalahan, data yang diterima akan disimpan di *expenses*.

Source code untuk filter *expenses* berdasarkan tanggal yang telah ditentukan oleh pengguna dengan button GO:

    return (

      <div>

<div className={classes.search}>

                <Typography variant="h6" className={classes.title}>Expenditures per category </Typography>

                <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DatePicker

                    disableFuture

                    format="dd/MM/yyyy"

                    label="FROM"

                    views={["year", "month", "date"]}

                    value={firstDay}

                    className={classes.textField}

                    onChange={handleDateChange('firstDay')}

                />

                <DatePicker

                    format="dd/MM/yyyy"

                    label="TO"

                    views={["year", "month", "date"]}

                    value={lastDay}

                    className={classes.textField}

                    onChange={handleDateChange('lastDay')}

                />

        </MuiPickersUtilsProvider>

        <Button variant="contained" color="secondary" onClick={searchClicked}>GO</Button>

        </div>

                <div style={{width: 550, margin: 'auto'}}>

                <svg viewBox="0 0 320 320">

            <VictoryPie standalone={false} data={expenses.monthAVG} innerRadius={50} theme={VictoryTheme.material}

                labelRadius={({ innerRadius }) => innerRadius + 14 }

                labelComponent={<VictoryLabel angle={0} style={[{

                    fontSize: '11px',

                    fill: '#0f0f0f'

                },

                {

                    fontSize: '10px',

                    fill: '#013157'

                }]} text={( {datum} ) => `${datum.x}\n $${datum.y}`}/>}

                 />

                 <VictoryLabel

          textAnchor="middle"

          style={{ fontSize: 14, fill: '#8b8b8b' }}

          x={175} y={170}

          text={`Spent \nper category`}

        />

        </svg>

                 </div>

        </div>

    )

  }

1. *Filter* *expenses*

    return (

      <div>

*return* menandakan bahwa komponen ini akan mengembalikan sesuatu untuk ditampilkan di layar. *<div>* adalah elemen *HTML* yang digunakan sebagai wadah (*container*)*.*

1. *Search div*

<div className={classes.search}>

                <Typography variant="h6" className={classes.title}>Expenditures per category </Typography>

                <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DatePicker

                    disableFuture

                    format="dd/MM/yyyy"

                    label="FROM"

                    views={["year", "month", "date"]}

                    value={firstDay}

                    className={classes.textField}

                    onChange={handleDateChange('firstDay')}

                />

                <DatePicker

                    format="dd/MM/yyyy"

                    label="TO"

                    views={["year", "month", "date"]}

                    value={lastDay}

                    className={classes.textField}

                    onChange={handleDateChange('lastDay')}

                />

</MuiPickersUtilsProvider>

        <Button variant="contained" color="secondary" onClick={searchClicked}>GO</Button>

        </div>

* *className={classes.search}* memberi gaya khusus dari *CSS* untuk elemen ini.
* *<Typography>* menampilkan teks "*Expenditures per category*" sebagai judul.
* *<MuiPickersUtilsProvider>* dan *<DatePicker>* adalah komponen dari pustaka *material-ui-pickers* untuk memilih tanggal.
* *firstDay* dan *lastDay* adalah nilai tanggal yang dipilih.
* *handleDateChange* adalah fungsi yang mengatur perubahan tanggal.
* *<Button>* adalah tombol yang memiliki teks "*GO*" dan memanggil fungsi *searchClicked* ketika diklik.

                <div style={{width: 550, margin: 'auto'}}>

                <svg viewBox="0 0 320 320">

            <VictoryPie standalone={false} data={expenses.monthAVG} innerRadius={50} theme={VictoryTheme.material}

                labelRadius={({ innerRadius }) => innerRadius + 14 }

                labelComponent={<VictoryLabel angle={0} style={[{

                    fontSize: '11px',

                    fill: '#0f0f0f'

                },

                {

                    fontSize: '10px',

                    fill: '#013157'

                }]} text={( {datum} ) => `${datum.x}\n $${datum.y}`}/>}

                 />

                 <VictoryLabel

          textAnchor="middle"

          style={{ fontSize: 14, fill: '#8b8b8b' }}

          x={175} y={170}

          text={`Spent \nper category`}

        />

        </svg>

                 </div>

        </div>

    )

  }

Bagian ini menampilkan grafik pie menggunakan pustaka *Victory*.

* + *expenses.monthAVG* adalah data yang digunakan untuk membuat grafik pie.
  + *innerRadius* menentukan ukuran radius dalam dari pie.
  + *VictoryTheme*.material memberikan tema khusus untuk grafik.
  + *labelComponent* mengatur tampilan *label* pada grafik.
  + *VictoryLabel* menampilkan teks *"Spent per category"* di tengah grafik.
  + *</div>* menutup elemen *<div>* utama dan menyelesaikan fungsi *return* komponen.

### J. Add Expense button

Source code yang membawa pengguna kepada form menambah expense baru:

{auth.isAuthenticated() && (

            <span>

              <Link to="/expenses/new">

                <Button style={isButtonActive(history, "/expenses/new")}>

                  <AddIcon style={{ marginRight: 4 }} /> Add Expense

                </Button>

              </Link>

            </span>

          )}

1. Autentikasi

{auth.isAuthenticated() && (

a*uth.isAuthenticated()* adalah metode yang memeriksa apakah pengguna sudah terautentikasi. *&&* adalah operator logika yang berarti "DAN". Jadi, jika *auth.isAuthenticated()* mengembalikan nilai *true*, maka bagian setelah *&&* akan dieksekusi dan ditampilkan.

1. *Span & Link To*

            <span>

              <Link to="/expenses/new">

*<span>* adalah elemen *HTML* yang digunakan sebagai wadah tanpa efek tata letak khusus. *<Link>* adalah komponen dari pustaka *react-router-dom* yang digunakan untuk navigasi antara halaman di aplikasi *React. to="/expenses/new"* menentukan bahwa ketika *link* diklik, pengguna akan diarahkan ke URL */expenses/new*.

1. *Button*

                <Button style={isButtonActive(history, "/expenses/new")}>

                  <AddIcon style={{ marginRight: 4 }} /> Add Expense

                </Button>

              </Link>

            </span>

          )}

* + *<Button>* adalah komponen tombol dari pustaka *UI* yang digunakan.
  + *style={isButtonActive(history, "/expenses/new")}* menambahkan gaya pada tombol. *isButtonActive* adalah fungsi yang mengembalikan objek gaya yang mengubah tampilan tombol berdasarkan apakah halaman saat ini adalah /expenses/new.
  + *<AddIcon style={{ marginRight: 4 }} />* adalah ikon yang ditampilkan di dalam tombol, dengan sedikit jarak di sebelah kanannya *(marginRight: 4)*.
  + *Add Expense* adalah teks yang ditampilkan di dalam tombol.

Source code yang menjalankan form tambah expense baru:

export default function NewExpense() {

  const classes = useStyles()

  const [values, setValues] = useState({

      title: '',

      category: '',

      amount: '',

      incurred\_on: new Date(),

      notes: '',

      error: ''

  })

  const jwt = auth.isAuthenticated()

  const handleChange = name => event => {

    setValues({...values, [name]: event.target.value })

  }

  const handleDateChange = date => {

    setValues({...values, incurred\_on: date })

  }

  const clickSubmit = () => {

    const expense = {

        title: values.title || undefined,

        category: values.category || undefined,

        amount: values.amount || undefined,

        incurred\_on: values.incurred\_on || undefined,

        notes: values.notes || undefined

    }

    create({

        t: jwt.token

    }, expense).then((data) => {

        if (data.error) {

          setValues({...values, error: data.error})

        } else {

          setValues({...values, error: '', redirect: true})

        }

    })

  }

    if (values.redirect) {

      return (<Redirect to={'/'}/>)

    }

    return (<div>

      <Card className={classes.card}>

        <CardContent>

          <Typography type="headline" component="h2" className={classes.title}>

            Expense Record

          </Typography>

          <br/>

          <TextField id="title" label="Title" className={classes.textField} value={values.title} onChange={handleChange('title')} margin="normal"/><br/>

          <TextField id="amount" label="Amount ($)" className={classes.textField} value={values.amount} onChange={handleChange('amount')} margin="normal" type="number"/><br/>

          <TextField id="category" label="Category" className={classes.textField} value={values.category} onChange={handleChange('category')} margin="normal"/><br/>

          <br/>

          <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DateTimePicker

                    label="Incurred on"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={values.incurred\_on}

                    onChange={handleDateChange}

                    showTodayButton

                />

          </MuiPickersUtilsProvider>

          <br/>

          <br/>

          <TextField

            id="multiline-flexible"

            label="Notes"

            multiline

            rows="2"

            value={values.notes}

            onChange={handleChange('notes')}

            className={classes.textField}

            margin="normal"

          /><br/> <br/>

           {

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}</Typography>)

          }

        </CardContent>

1. *NewExpense function*

export default function NewExpense() {

*export default function NewExpense()* mengekspor komponen *NewExpense* sebagai default dari modul ini, sehingga bisa diimpor dan digunakan di bagian lain dari aplikasi.

1. Mendefinisikan Kelas dan State

  const classes = useStyles()

  const [values, setValues] = useState({

      title: '',

      category: '',

      amount: '',

      incurred\_on: new Date(),

      notes: '',

      error: ''

  })

  const jwt = auth.isAuthenticated()

* + *const classes = useStyles()* menginisialisasi gaya yang digunakan dalam komponen ini.
  + *const [values, setValues] = useState({...})* mendefinisikan state lokal menggunakan useState untuk menyimpan nilai-nilai dari form yang digunakan untuk membuat pengeluaran baru.
  + *const jwt = auth.isAuthenticated()* mengambil informasi autentikasi dari pengguna saat ini.

1. *handleChange*

  const handleChange = name => event => {

    setValues({...values, [name]: event.target.value })

  }

  const handleDateChange = date => {

    setValues({...values, incurred\_on: date })

  }

* + *handleChange* adalah fungsi yang mengupdate state ketika ada perubahan di input form.
  + *handleDateChange* adalah fungsi yang mengupdate tanggal incurred\_on ketika ada perubahan di picker tanggal.

1. *clickSubmit*

  const clickSubmit = () => {

    const expense = {

        title: values.title || undefined,

        category: values.category || undefined,

        amount: values.amount || undefined,

        incurred\_on: values.incurred\_on || undefined,

        notes: values.notes || undefined

    }

    create({

        t: jwt.token

    }, expense).then((data) => {

        if (data.error) {

          setValues({...values, error: data.error})

        } else {

          setValues({...values, error: '', redirect: true})

        }

    })

  }

*clickSubmit* adalah fungsi yang mengumpulkan data dari *state* dan mengirimkannya ke server melalui fungsi *create*. Jika server mengembalikan *error*, *error* tersebut disimpan dalam state. Jika sukses, *state* *redirect* diatur menjadi true.

1. *Redirection*

    if (values.redirect) {

      return (<Redirect to={'/'}/>)

    }

Jika *values.redirect* bernilai *true*, komponen akan melakukan redirect ke halaman utama *('/').*

1. Tampilkan komponen

    return (<div>

      <Card className={classes.card}>

        <CardContent>

          <Typography type="headline" component="h2" className={classes.title}>

            Expense Record

          </Typography>

          <br/>

          <TextField id="title" label="Title" className={classes.textField} value={values.title} onChange={handleChange('title')} margin="normal"/><br/>

          <TextField id="amount" label="Amount ($)" className={classes.textField} value={values.amount} onChange={handleChange('amount')} margin="normal" type="number"/><br/>

          <TextField id="category" label="Category" className={classes.textField} value={values.category} onChange={handleChange('category')} margin="normal"/><br/>

          <br/>

          <MuiPickersUtilsProvider utils={DateFnsUtils}>

                <DateTimePicker

                    label="Incurred on"

                    className={classes.textField}

                    views={["year", "month", "date"]}

                    value={values.incurred\_on}

                    onChange={handleDateChange}

                    showTodayButton

                />

          </MuiPickersUtilsProvider>

          <br/>

          <br/>

          <TextField

            id="multiline-flexible"

            label="Notes"

            multiline

            rows="2"

            value={values.notes}

            onChange={handleChange('notes')}

            className={classes.textField}

            margin="normal"

          /><br/> <br/>

           {

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}</Typography>)

          }

        </CardContent>

Mengembalikan JSX yang akan ditampilkan oleh komponen ini.

* Terdapat elemen form seperti *TextField* untuk memasukkan judul, jumlah, kategori, dan catatan *expense*.
* *DateTimePicker* digunakan untuk memilih tanggal *expense*.
* *Error* message ditampilkan jika ada *error* dalam pengiriman data.

Source code untuk button Submit dan Cancel

        <CardActions>

          <Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>

          <Link to='/myauctions' className={classes.submit}><Button variant="contained">Cancel</Button></Link>

        </CardActions>

      </Card>

    </div>)

1. *CardActions*

        <CardActions>

*<CardActions>* adalah komponen dari *Material-UI* yang digunakan untuk menampung aksi-aksi yang berkaitan dengan kartu tersebut, biasanya berupa tombol.

1. *Button* *Sumit*

          <Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>

*<Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>* adalah tombol yang memiliki warna utama (biru) dan gaya *'contained'* (tombol yang solid). Ketika tombol ini diklik, fungsi *clickSubmit* akan dipanggil untuk mengirim data pengeluaran baru ke server.

1. *Link & Cancel Button*

          <Link to='/myauctions' className={classes.submit}><Button variant="contained">Cancel</Button></Link>

        </CardActions>

      </Card>

    </div>)

*<Link to='/myauctions' className={classes.submit}><Button variant="contained">Cancel</Button></Link>* adalah link yang mengarahkan pengguna ke halaman */myauctions* jika pengguna memilih untuk membatalkan pembuatan pengeluaran baru. Tombol *'Cancel'* ini juga memiliki gaya 'contained'.

### J. My Profile Menu

##### Edit profile button

Source code yang membawa user ke tampilan edit profile:

<Link to={"/user/edit/" + user.\_id}>

                  <IconButton aria-label="Edit" color="primary">

                    <Edit/>

                  </IconButton>

                </Link>

1. *Link*

<Link to={"/user/edit/" + user.\_id}>

Bagian ini membuat sebuah tautan (*link*) yang akan mengarahkan pengguna ke halaman edit. *URL* tujuan akan memiliki format */user/edit/* diikuti dengan *ID* pengguna *(user.\_id)*. Jadi, jika *ID* pengguna adalah *123*, maka tautannya akan menjadi */user/edit/123*.

1. Button

                  <IconButton aria-label="Edit" color="primary">

                    <Edit/>

                  </IconButton>

                </Link>

Di dalam tautan ini, ada sebuah tombol dengan ikon.

* + *IconButton* adalah komponen dari *Material-UI,* sebuah pustaka komponen untuk *React* yang memudahkan pembuatan antarmuka yang menarik.
  + *aria-label="Edit"* adalah atribut yang memberikan deskripsi teks untuk tombol ini bagi pengguna yang menggunakan pembaca layar, membantu aksesibilitas.
  + *color="primary"* mengatur warna tombol menjadi warna utama yang sudah ditentukan oleh tema *Material-UI* (biasanya biru).
  + Di dalam *IconButton*, ada komponen *Edit* yang merupakan ikon berbentuk pensil atau ikon edit lainnya. Ini juga dari *Material-UI*.
  + *</IconButton>* menutup komponen *IconButton*.
  + *</Link>* menutup komponen Link.

Source code yang menjalankan tampilan edit profile yang pengguna lihat:

export default function EditProfile({ match }) {

  const classes = useStyles()

  const [values, setValues] = useState({

    name: '',

    password: '',

    email: '',

    open: false,

    error: '',

    redirectToProfile: false

  })

  const jwt = auth.isAuthenticated()

  useEffect(() => {

    const abortController = new AbortController()

    const signal = abortController.signal

    read({

      userId: match.params.userId

    }, {t: jwt.token}, signal).then((data) => {

      if (data && data.error) {

        setValues({...values, error: data.error})

      } else {

        setValues({...values, name: data.name, email: data.email})

      }

    })

    return function cleanup(){

      abortController.abort()

    }

  }, [match.params.userId])

  const clickSubmit = () => {

    const user = {

      name: values.name || undefined,

      email: values.email || undefined,

      password: values.password || undefined

    }

    update({

      userId: match.params.userId

    }, {

      t: jwt.token

    }, user).then((data) => {

      if (data && data.error) {

        setValues({...values, error: data.error})

      } else {

        setValues({...values, userId: data.\_id, redirectToProfile: true})

      }

    })

  }

  const handleChange = name => event => {

    setValues({...values, [name]: event.target.value})

  }

    if (values.redirectToProfile) {

      return (<Redirect to={'/user/' + values.userId}/>)

    }

    return (

      <Card className={classes.card}>

        <CardContent>

          <Typography variant="h6" className={classes.title}>

            Edit Profile

          </Typography>

          <TextField id="name" label="Name" className={classes.textField} value={values.name} onChange={handleChange('name')} margin="normal"/><br/>

          <TextField id="email" type="email" label="Email" className={classes.textField} value={values.email} onChange={handleChange('email')} margin="normal"/><br/>

          <TextField id="password" type="password" label="Password" className={classes.textField} value={values.password} onChange={handleChange('password')} margin="normal"/>

          <br/> {

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}

            </Typography>)

          }

        </CardContent>

1. *EditProfile function*

export default function EditProfile({ match }) {

  const classes = useStyles()

*EditProfile* adalah fungsi komponen *React* yang menerima prop match. *useStyles* adalah hook untuk mengambil kelas *CSS* yang digunakan untuk styling.

1. *useState*

  const [values, setValues] = useState({

    name: '',

    password: '',

    email: '',

    open: false,

    error: '',

    redirectToProfile: false

  })

*useState* digunakan untuk mendefinisikan *state* lokal *values* yang berisi data profil pengguna seperti *name*, *password*, *email*, dan lain-lain.

1. *Authenticate*

  const jwt = auth.isAuthenticated()

*jwt* menyimpan token autentikasi yang diambil dari metode *isAuthenticated* di objek *auth*.

1. *useEffect*

  useEffect(() => {

    const abortController = new AbortController()

    const signal = abortController.signal

    read({

      userId: match.params.userId

    }, {t: jwt.token}, signal).then((data) => {

      if (data && data.error) {

        setValues({...values, error: data.error})

      } else {

        setValues({...values, name: data.name, email: data.email})

      }

    })

    return function cleanup(){

      abortController.abort()

    }

  }, [match.params.userId])

* + *useEffect* digunakan untuk mengambil data pengguna ketika komponen pertama kali dirender atau userId dalam match.params berubah.
  + *abortController* digunakan untuk membatalkan permintaan jika komponen di-unmount sebelum permintaan selesai.
  + *read* adalah fungsi untuk mengambil data pengguna dari server.
  + Data yang diambil akan disimpan dalam state *values*.

1. *Submit Button*

  const clickSubmit = () => {

    const user = {

      name: values.name || undefined,

      email: values.email || undefined,

      password: values.password || undefined

    }

    update({

      userId: match.params.userId

    }, {

      t: jwt.token

    }, user).then((data) => {

      if (data && data.error) {

        setValues({...values, error: data.error})

      } else {

        setValues({...values, userId: data.\_id, redirectToProfile: true})

      }

    })

  }

* + *clickSubmit* adalah fungsi yang dipanggil ketika tombol submit diklik.
  + *update* adalah fungsi untuk mengirim data yang diperbarui ke server.
  + Jika *update* berhasil, state *redirectToProfile* diubah menjadi true sehingga pengguna akan diarahkan ke halaman profil.

1. *handleChange*

  const handleChange = name => event => {

    setValues({...values, [name]: event.target.value})

  }

*handleChange* adalah fungsi untuk mengupdate state values ketika input berubah.

1. Komponen *EditProfile*

    if (values.redirectToProfile) {

      return (<Redirect to={'/user/' + values.userId}/>)

    }

    return (

      <Card className={classes.card}>

        <CardContent>

          <Typography variant="h6" className={classes.title}>

            Edit Profile

          </Typography>

          <TextField id="name" label="Name" className={classes.textField} value={values.name} onChange={handleChange('name')} margin="normal"/><br/>

          <TextField id="email" type="email" label="Email" className={classes.textField} value={values.email} onChange={handleChange('email')} margin="normal"/><br/>

          <TextField id="password" type="password" label="Password" className={classes.textField} value={values.password} onChange={handleChange('password')} margin="normal"/>

          <br/> {

            values.error && (<Typography component="p" color="error">

              <Icon color="error" className={classes.error}>error</Icon>

              {values.error}

            </Typography>)

          }

        </CardContent>

* Jika *redirectToProfile* bernilai true, pengguna akan diarahkan ke halaman profil.
* Komponen ini merender form untuk mengedit profil pengguna menggunakan beberapa komponen *Material-UI* seperti *Card, CardContent, Typography,* dan *TextField*.
* *Error* ditampilkan jika ada kesalahan dalam proses update.

Source code untuk button Submit pada Edit Profile:

        <CardActions>

          <Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>

        </CardActions>

      </Card>

    )

}

1. *CardActions*

        <CardActions>

*CardActions* adalah komponen dari *Material-UI* yang digunakan untuk menampung elemen-elemen aksi, seperti tombol, yang biasanya diletakkan di bagian bawah kartu.

1. *Button*

          <Button color="primary" variant="contained" onClick={clickSubmit} className={classes.submit}>Submit</Button>

        </CardActions>

      </Card>

    )

}

* + *Button*: Tombol ini berwarna biru (*primary*), memiliki gaya *contained* (terlihat seperti tombol *solid*), dan akan memanggil fungsi *clickSubmit* ketika diklik.
  + *className: className={classes.submit}* digunakan untuk menambahkan kelas CSS khusus untuk tombol ini agar bisa di-*styling* sesuai keinginan.

##### Delete profile button

Source code untuk icon delete profile / akun:

<DeleteUser userId={user.\_id}/>

Panggilan *<DeleteUser userId={user.\_id}/>* digunakan untuk memasukkan instance dari komponen DeleteUser ke dalam suatu halaman atau komponen yang lebih besar. Dalam hal ini, properti userId diberikan dengan nilai \_id dari objek user.

Source code untuk delete account / profile

export default function DeleteUser(props) {

  const [open, setOpen] = useState(false)

  const [redirect, setRedirect] = useState(false)

  const jwt = auth.isAuthenticated()

  const clickButton = () => {

    setOpen(true)

  }

  const deleteAccount = () => {

    remove({

      userId: props.userId

    }, {t: jwt.token}).then((data) => {

      if (data && data.error) {

        console.log(data.error)

      } else {

        auth.clearJWT(() => console.log('deleted'))

        setRedirect(true)

      }

    })

  }

  const handleRequestClose = () => {

    setOpen(false)

  }

  if (redirect) {

    return <Redirect to='/'/>

  }

    return (<span>

      <IconButton aria-label="Delete" onClick={clickButton} color="secondary">

        <DeleteIcon/>

      </IconButton>

1. *DeleteUser function*

export default function DeleteUser(props) {

Fungsi yang diekspor sebagai *default* dari modul *JavaScript*. Dalam hal ini, komponen bernama *DeleteUser* diekspor secara *default*.

1. *useState*

  const [open, setOpen] = useState(false)

  const [redirect, setRedirect] = useState(false)

*Hook* yang digunakan dalam React untuk mengelola state pada komponen fungsional. Di sini, ada dua *state* yang dikelola: *open* dan *redirect*. *open* digunakan untuk menentukan apakah dialog konfirmasi untuk menghapus akun sedang terbuka atau tidak, sedangkan *redirect* menentukan apakah pengguna akan diarahkan ke halaman lain setelah menghapus akun.

1. *jwt*

  const jwt = auth.isAuthenticated()

JWT digunakan untuk mengautentikasi pengguna yang melakukan tindakan menghapus akun.

1. *clickButton*

  const clickButton = () => {

    setOpen(true)

  }

Fungsi yang dipanggil saat tombol hapus di-klik. Fungsi ini mengatur *state* *open* menjadi *true*, yang akan membuka dialog konfirmasi.

1. *deleteAccount*

  const deleteAccount = () => {

    remove({

      userId: props.userId

    }, {t: jwt.token}).then((data) => {

      if (data && data.error) {

        console.log(data.error)

      } else {

        auth.clearJWT(() => console.log('deleted'))

        setRedirect(true)

      }

    })

  }

Fungsi yang dipanggil ketika pengguna mengonfirmasi penghapusan akun mereka. Ini membuat permintaan untuk menghapus akun menggunakan fungsi *remove*, yang kemudian menghapus token *JWT* dari penyimpanan lokal pengguna jika permintaan berhasil.

1. *handleRequestClose*

  const handleRequestClose = () => {

    setOpen(false)

  }

  if (redirect) {

    return <Redirect to='/'/>

  }

Fungsi yang dipanggil ketika dialog konfirmasi ditutup. Ini mengatur *state* *open* kembali ke *false*, menutup *dialog.*

1. *Komponen*

    return (<span>

      <IconButton aria-label="Delete" onClick={clickButton} color="secondary">

        <DeleteIcon/>

      </IconButton>

Komponen ini akan merender ikon tombol hapus (*<DeleteIcon />)* yang dapat diklik. Ketika tombol ini diklik, itu akan memicu fungsi *clickButton*, yang akan membuka dialog konfirmasi. Jika *redirect* diatur ke *true*, maka pengguna akan dialihkan ke halaman utama menggunakan komponen *<Redirect to='/'/>*.

Source code popup dialog untuk konfirmasi lagi apakah akun profile ingin dihapus:

      <Dialog open={open} onClose={handleRequestClose}>

        <DialogTitle>{"Delete Account"}</DialogTitle>

        <DialogContent>

          <DialogContentText>

            Confirm to delete your account.

          </DialogContentText>

        </DialogContent>

1. *Dialog*

      <Dialog open={open} onClose={handleRequestClose}>

Komponen *dialog* dari *material-ui* yang akan muncul saat *state* *open* bernilai *true*. Properti *onClose* mengatur fungsi yang akan dipanggil saat *dialog* ditutup, dalam hal ini, itu memanggil fungsi *handleRequestClose*.

        <DialogTitle>{"Delete Account"}</DialogTitle>

Judul dialog yang akan ditampilkan kepada pengguna, dalam hal ini, judulnya adalah "*Delete Account*".

        <DialogContent>

          <DialogContentText>

            Confirm to delete your account.

          </DialogContentText>

        </DialogContent>

*<DialogContent>* adalah komponen yang berisi konten *dialog*. *<DialogContentText>* adalah komponen yang digunakan untuk menampilkan teks dalam *dialog*. Di sini, teksnya adalah "*Confirm to delete your account.*"

Source code untuk button Cancel dan button Confirm:

1. *DialogActions*

        <DialogActions>

*<DialogActions>* adalah komponen yang digunakan untuk menampilkan aksi di bagian bawah dialog.

1. Button *onClick handleRequestClose*

          <Button onClick={handleRequestClose} color="primary">

            Cancel

          </Button>

*<Button onClick={handleRequestClose} color="primary">Cancel</Button>* adalah tombol untuk membatalkan operasi penghapusan akun. Ketika tombol ini diklik, itu akan memanggil fungsi *handleRequestClose* untuk menutup dialog.

1. *Button onClick deleteAccount*

          <Button onClick={deleteAccount} color="secondary" autoFocus="autoFocus">

            Confirm

          </Button>

        </DialogActions>

      </Dialog>

    </span>)

}

*<Button onClick={deleteAccount} color="secondary" autoFocus="autoFocus">Confirm</Button>* adalah tombol untuk mengonfirmasi penghapusan akun. Ketika tombol ini diklik, itu akan memanggil fungsi deleteAccount yang akan memulai proses penghapusan akun.

1. *DeleteUser*.*propTypes*

DeleteUser.propTypes = {

  userId: PropTypes.string.isRequired

}

*DeleteUser.propTypes = { userId: PropTypes.string.isRequired }* akan mendefinisikan properti yang diterima oleh komponen *DeleteUser*. Properti userId harus berupa string dan wajib.

# BAB IV

# SIMPULAN

Dalam pengembangan aplikasi media sosial dan tracker expenses, penggunaan MERN stack (MongoDB, Express.js, React.js, dan Node.js) telah terbukti sangat efektif. Platform ini memberikan fleksibilitas yang besar dalam pengembangan aplikasi web, memungkinkan kami untuk membangun dua aplikasi yang responsif dan dinamis. MongoDB digunakan sebagai basis data untuk menyimpan informasi pengguna, postingan, serta data keuangan untuk tracker expenses, dengan struktur data yang fleksibel. Express.js berperan sebagai framework back-end untuk mengelola permintaan HTTP dan routing. Sementara itu, Node.js sebagai runtime environment JavaScript memungkinkan kami untuk menjalankan kode server-side. Di sisi front-end, React.js digunakan untuk membangun antarmuka pengguna yang interaktif dan mudah digunakan untuk kedua aplikasi. Desain UI/UX yang diterapkan memberikan pengalaman pengguna yang intuitif dan menyenangkan. Melalui penggunaan MERN stack, kami berhasil mengembangkan aplikasi yang memenuhi kebutuhan pengguna dalam hal jejaring sosial dan pengelolaan keuangan pribadi dengan efisien.

Pemahaman penggunaan warna untuk design pada aplikasi ini adalah warna dapat digunakan untuk menonjolkan elemen penting dalam aplikasi, seperti tombol, menu, dan informasi utama. Hal ini membantu pengguna dalam memahami dan menggunakan aplikasi dengan lebih mudah.

Warna juga dapat digunakan untuk menciptakan suasana hati dan identitas merek yang sesuai dengan tujuan aplikasi. Penggunaan warna yang kontras dan jelas dapat membantu pengguna dengan gangguan penglihatan dalam menggunakan aplikasi.

Penggunaan ikon yang benar dan mudah dimengerti membantu pengguna untuk membantu pengguna memahami fungsi atau fitur aplikasi dengan cepat tanpa harus membaca teks. Hal ini penting dalam meningkatkan efisiensi dan kemudahan penggunaan aplikasi. Penggunaan ikon yang konsisten dan standar dalam aplikasi juga dapat memudahkan pengguna yang sudah terbiasa dengan ikon tersebut dalam aplikasi lain. Ikon juga dapat digunakan sebagai elemen navigasi membantu pengguna menemukan fitur atau informasi yang mereka cari dengan mudah.

Dengan ini penggunaan MERN dalam membuat aplikasi social media dan expenses tracker ini dapat memberikan penegtahuan kepada kita mengenai cara, penegrtian dari MERN dan aplikasi dari sosail media dan expenses tracker.

# DAFTAR PUSTAKA

Abdur Ra’uf Al Farras1, A. D. A. A. F. A. P. Y. F. Z. D. H. S. H. P. L. P. I. B. R. H. A. Y. A. P. (2023). *Pelatihan HTML Dan CSS Dasar Menggunakan Visual Studio CodeDi SMK Setia Bhakti*. *1*, 200–204.

Andrew Mead. (2018). *Get this book in print  Front Cover  Learning Node.js Development*. Packt Publishing Ltd.

Annette Chacko. (2023, November 7). *Your guide to social media comments: How to post and respond*. SPROUT SOCIAL.

Asst.Prof. Abhishek Dadhich, S. J. S. J. S. M. (2023). Expense Tracker . *INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR)*, *10*(2).

Basarat Syed. (n.d.). *Beginning Node.js*. Apress.

Chakradhar Avinash Devarapalli. (2020). Employing Build Tools: Optimizing Frontend Development with Webpack, Gulp, or Grunt. *Journal of Technological Innovation*, *1*(1).

D91 Labs. (2020). Personal Finance Survey 2020-Expenses, Savings & Investments. In *D91 Labs*.

Mohammad Shahanewaz Shahabuddin. (2023). *Creating an E-commerce Web Application for Student Using MERN Stack.*

Mohanish Bawane1, I. G. V. J. R. N. Prof. S. A. B. (2022). A Review on Technologies used in MERN stack. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, *10*(1 Jan 2022), 2321–9653.

Monika Mehra, M. K. A. M. C. S. S. (2021). *MERN Stack Web Development*. *25*(6).

Muhamad Ayub, S. F. S. (n.d.). DAMPAK SOSIAL MEDIA TERHADAP INTERAKSI SOSIAL PADA  REMAJA: KAJIAN SISTEMATIK . *Jurnal Penelitian Bimbingan Dan Konseling Vol 7* , *7*.

Muhammad Iqbal Sultan. (2020). Efektifitas Penggunaan Fitur Instagram Dalam  Meningkatkan Pertemanan Remaja SMA Negeri 1 Maros Di Era Digital . *Jurnal Ilmu Komunikasi*, *08*(02), 178–190.

Nidhi Jitendra Jadhav, R. V. C. T. M. G. Damayanti. D. P. (2022). EXPENSE TRACKER . *International  Research  Journal of  Modernization in Engineering  Technology and  Science* , *4*.

Ojaswi Awasthi. (n.d.). *Mern Stack: Chatting App in React and NOSQL*.

Rully Pramudita, R. W. A. A. N. A. N. S. S. A. (2021). *PENGGUNAAN APLIKASI FIGMA DALAM MEMBANGUNUI/UX YANG INTERAKTIF PADA PROGRAM STUDI TEKNIK INFORMATIKA STMIK TASIKMALAYA*. *3*(1).

Shama Hoque. (2018). *Full-Stack React Projects: Learn MERN stack development by building modern*. Packt Publishing Ltd.

Tessalonica Putry Avrylya, Y. A. S. (2024). Comparison of Search Response Time Using Text Indexing on MongoDB and ArangoDB Web-based. *MALCOM: Indonesian Journal OfMachine Learningand Computer Science*, *4*, 777–785.

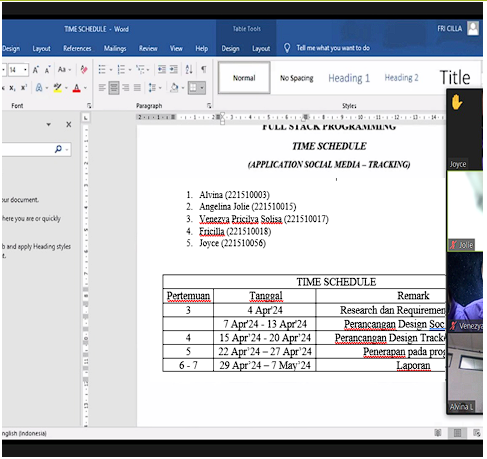
We Are Social Digital 2023. (n.d.). *DIGITAL 2023*. We Are Social .

# LAMPIRAN

Link Github: <https://github.com/alvina893/TM_221510003_FullStack_MERN_>

Desain Figma *Social Media:* [*https://www.figma.com/design/dtI4TDlsEGpiefJJpp0XYq/SOSIAL-MEDIA?node-id=0-1*](https://www.figma.com/design/dtI4TDlsEGpiefJJpp0XYq/SOSIAL-MEDIA?node-id=0-1)

Desain Figma *Expense Tracker:* [*https://www.figma.com/design/cII9n3UoFFzyIhX0qbbIaK/Untitled?node-id=0-1*](https://www.figma.com/design/cII9n3UoFFzyIhX0qbbIaK/Untitled?node-id=0-1)

****

****

